

Universitatea „Dunărea de Jos” din Galați
Școala Doctorală de Științe Fundamentale și Inginerești



TEZĂ DE DOCTORAT

- REZUMAT -

**Contribuții privind monitorizarea și
securitatea rețelelor de calculatoare
— sisteme de stocare partajate în rețele de
calculatoare**

**Doctorand,
Adrian Istrate**

**Conducător științific,
Prof. univ. dr. ing. Luminița DUMITRIU**

**Seria I 2: Calculatoare și tehnologia informației nr. 11
Galați
2025**

Universitatea „Dunărea de Jos” din Galați
Școala Doctorală de Științe Fundamentale și Inginerești



TEZĂ DE DOCTORAT

- REZUMAT -

Contribuții privind monitorizarea și securitatea rețelelor de calculatoare - sisteme de stocare partajate în rețele de calculatoare

Doctorand,
Adrian Istrate

Președinte:	Prof. univ. dr. ing. habil. BARBU Marian Rector al Universității „Dunărea de Jos” din Galați
Conducător științific:	Prof. univ. dr. ing. DUMITRIU Luminița Universitatea „Dunărea de Jos” din Galați
Referenți științifici:	Prof. univ. dr. ing. habil. RUGHINIȘ Răzvan-Victor Universitatea Națională de Știință și Tehnologie POLITEHNICA București Conf. univ. dr. ing. CEBUC Emil-Ioan Universitatea Tehnică din Cluj-Napoca Conf. univ. dr. ing. COCU Adina Universitatea „Dunărea de Jos” din Galați

Seria I 2: Calculatoare și tehnologia informației nr. 11
Galați
2025

Seriile tezelor de doctorat susținute public în UDJG începând cu 1 octombrie 2013 sunt:

Domeniul fundamental **ȘTIINȚE INGINEREȘTI**

- Seria I 1: **Biotehnologii**
- Seria I 2: **Calculatoare și tehnologia informației**
- Seria I 3: **Inginerie electrică**
- Seria I 4: **Inginerie industrială**
- Seria I 5: **Ingineria materialelor**
- Seria I 6: **Inginerie mecanică**
- Seria I 7: **Ingineria produselor alimentare**
- Seria I 8: **Ingineria sistemelor**
- Seria I 9: **Inginerie și management în agricultură și dezvoltare rurală**

Domeniul fundamental **ȘTIINȚE SOCIALE**

- Seria E 1: **Economie**
- Seria E 2: **Management**
- Seria E 3: **Marketing**
- Seria SSEF: **Știința sportului și educației fizice**
- Seria SJ: **Drept**

Domeniul fundamental **ȘTIINȚE UMANISTE**

- Seria U 1: **Filologie - Engleză**
- Seria U 2: **Filologie - Română**
- Seria U 3: **Istorie**
- Seria U 4: **Filologie - Franceză**

Domeniul fundamental **MATEMATICĂ ȘI ȘTIINȚE ALE NATURII**

- Seria C: **Chimie**

Domeniul fundamental **ȘTIINȚE BIOLOGICE ȘI BIOMEDICALE**

- Seria M: **Medicină**
- Seria F: **Farmacie**

Cuprins

Cuprins	i
Introducere	1
Obiective urmărite în cadrul lucrării	1
Conținutul capitolelor	2
Capitolul I - Arhitecturi pentru cloud computing și sisteme de stocare partajate - stadiul actual	5
1.1. Arhitecturi pentru cloud computing	5
1.1.1. Beneficiile cloud computing-ului în procesele educaționale	5
1.1.2. Modele de arhitecturi pentru cloud	6
1.2. Sisteme de stocare partajate pentru cloud computing	8
1.2.1. Stocarea la nivel de blocuri	9
1.2.2. Stocarea bazată pe fișiere	11
1.2.3. Sisteme de fișiere avansate și în rețea	13
1.2.4. Rețele pentru sisteme de stocare (SAN)	16
1.3. Tehnologii de rețea în arhitecturi de stocare performante	18
1.3.1. RDMA - Principii și mecanisme	18
1.3.2. RoCE și iWARP - Protocoale RDMA peste Ethernet și TCP	19
1.3.3. DCB - Data Center Bridging pentru rețele fără pierderi de pachete	20
1.3.4. Integrarea RDMA în protocoalele de stocare	21
1.4. Considerații de securitate	22
1.4.1. Elemente de securitate la nivelul stocării	22
1.4.2. Considerații de securitate în rețele RDMA	24
1.5. Concluzii intermediare	25
Capitolul II - Platforme cloud open-source pentru infrastructuri virtualizate	26
2.1. OpenStack	26
2.2. CloudStack	27
2.3. OpenNebula	27
2.4. Diferențe și similitudini între OpenStack, CloudStack și OpenNebula	28
2.4.1. Arhitectura și modelul de operare	29
2.4.2. Suportul pentru hipervizoare	30
2.4.3. Managementul sistemelor de stocare	30
2.4.4. Conectivitatea și serviciile de rețea	31
2.4.5. Planificatorul (scheduler) și alocarea resurselor	32
2.4.6. Managementul utilizatorilor și securitatea	32
2.4.7. Orchestrarea și automatizarea	33
2.4.8. Telemetria și monitorizarea	34
2.4.9. Interfața de management	34
2.4.10. Edge computing și cloud hibrid	35
2.4.11. Maturitate, dezvoltare și comunitate	36
2.4.12. Concluzii privind comparația platformelor de virtualizare	36
Capitolul III - Propunere de arhitectură experimentală pentru determinarea tiparelor de intrare/ieșire	38
3.1. Motivația testelor experimentale	38
3.2. Arhitectură propusă pentru testări	39
3.2.1. Arhitectura logică	39
3.2.2. Arhitectura hardware	39
3.2.3. Componentele arhitecturii pentru realizarea experimentelor	40
3.3. Alegerea mediului de stocare și a platformei de virtualizare	43
3.3.1. Sisteme de stocare în memoria volatilă	43
3.3.2. Alegerea sistemului de stocare folosit pentru experimente	44

3.3.3.	Instrumentarea monitorizării memoriei volatile partajate	45
3.3.4.	Alegerea platformei pentru managementul cloud-ului	46
3.3.5.	Configurarea și optimizarea sistemelor	47
3.4.	Concluzii intermediare	48
Capitolul IV - Metodologie pentru realizarea testelor experimentale		49
4.1.	Justificarea necesității testării experimentale	49
4.2.	Structura testelor și modalități de testare	50
4.2.1.	Configurația mașinilor virtuale	50
4.2.2.	Moduri de testare	50
4.2.3.	Organizarea seriilor de teste	50
4.2.4.	Fluxul experimental	51
4.3.	Mărimi măsurate și metrici de performanță	51
4.3.1.	Măsurători temporale la nivelul mașinilor virtuale	51
4.3.2.	Metrici de performanță la nivelul hipervizoarelor	51
4.3.3.	Măsurători la nivelul subsistemului de stocare	52
4.3.4.	Monitorizarea resurselor hardware	52
4.3.5.	Sincronizarea temporală	52
4.3.6.	Diagrame la nivel de set de teste (agregare pe rulări)	52
4.3.7.	Diagrame la nivel de rulare (diagnostic și încadrare)	53
4.4.	Metodologie pentru determinarea profilurilor de intrare/ieșire	53
4.4.1.	Cadrul general al mărimilor măsurate și convenții de notare	53
4.4.2.	Aproximarea liniară a repartițiilor blocurilor la nivelul sistemului de stocare ..	54
4.4.3.	Estimarea repartițiilor prin minimizarea entropiei relative	54
4.5.	Metodologie pentru evaluarea performanțelor sistemelor de stocare	55
4.5.1.	Configurații ale sistemelor de stocare partajate	55
4.5.2.	Definirea mărimilor măsurabile	56
4.6.	Concluzii intermediare	57
Capitolul V - Analiza rezultatelor experimentale		58
5.1.	Validarea preliminară a infrastructurii de monitorizare	58
5.1.1.	Test pentru validarea preliminară	58
5.1.2.	Diferențele integrale ale volumelor transferate	58
5.1.3.	Diferențele instantanee ale volumelor transferate	59
5.1.4.	Concluzii intermediare privind testul de validare	59
5.2.	Determinarea profilurilor de intrare/ieșire	59
5.2.1.	Scenarii de încărcare medie	60
5.2.2.	Scenarii de încărcare maximă	60
5.2.3.	Analiza profilurilor de intrare/ieșire determinate	61
5.3.	Investigații experimentale asupra sistemelor partajate de stocare	62
5.3.1.	Definirea cadrului experimental	62
5.3.2.	Sistem de fișiere ext4 cu RAID software - configurare și rezultate	62
5.3.3.	Sistem de fișiere XFS cu RAID software - configurare și rezultate	64
5.3.4.	Sistem ZFS - configurare și rezultate	65
5.3.5.	Sistem OCFS2 cu iSCSI - configurare și rezultate	67
5.4.	Concluzii experimentale	68
Capitolul VI - Concluzii generale, contribuții și direcții viitoare		69
6.1.	Concluzii generale	69
6.2.	Contribuții științifice	69
6.3.	Direcții viitoare de cercetare și dezvoltare	70
Referințe bibliografice		71
ANEXE		82

Introducere

Virtualizarea a transformat infrastructurile informatice din sisteme dedicate și greu extensibile în platforme elastice, distribuite, care deservește simultan numeroși utilizatori și aplicații. Ea a facilitat utilizarea eficientă a resurselor și modele operaționale de tip cloud computing, cu consum „la cerere”, abstractizare ridicată și mecanisme avansate de management. Beneficiile se manifestă în educație (laboratoare virtuale accesibile), în industrie (replicare rapidă a scenariilor pentru dezvoltare și testare, cu costuri reduse) și în domenii critice (financiar, medical), unde asigură servicii continue și scalabile.

În acest context, performanța devine cerință centrală: platforma trebuie să ofere timpi de răspuns previzibili, să susțină sarcini concurente și să mențină stabilitatea accesului la resurse. Performanța nu se reduce la frecvențe de procesor sau capacitate RAM, ci include latențe, lățime de bandă, număr de operații de intrare/ieșire pe secundă, stabilitate sub sarcină și reziliență. Subsistemul de stocare este critic: accesul simultan al mai multor mașini virtuale generează tipare eterogene de citire/scriere, cu dimensiuni de bloc variate și grade diferite de concurență, iar limitările sale pot compromite performanța globală indiferent de puterea de calcul. Complexitatea provine din diversitatea mediilor (locale/distribuite, HDD/SSD/hibride), a organizărilor (RAID), a protocoalelor orientate bloc sau fișier și a sistemelor de fișiere (ext4, XFS, ZFS), plus caching și jurnalizare.

Această realitate expune limitele testelor sintetice standardizate, utile pentru comparații generale, dar adesea nereprezentative pentru sarcini reale. Devine necesară profilarea I/O: proporții citiri/scrieri, distribuții ale dimensiunilor de bloc, gradul de paralelism și dinamica temporală. Analiza trebuie să cuprindă atât nivelul de bloc, cât și pe cel de fișier; interacțiunile dintre straturi (RAID, politici de alocare și jurnalizare) pot amplifica sau diminua performanța, iar optimizarea izolată este, de regulă, insuficientă. Rețeaua, hipervizoarele, discurile virtuale și protocoalele specifice adaugă puncte de blocaj suplimentare; alegerea protocoalelor (iSCSI, Fibre Channel, NFS, SMB) influențează latența, debitul și consistența.

Lucrarea propune o metodologie de evaluare bazată pe profiluri reale, nu pe modele simplificate. Monitorizarea colectează date corelate în timp, din mai multe puncte, cu un set redus, dar coerent de metrici. Pentru a separa efectele rețelei și ale infrastructurii de limitările mediilor persistente, se utilizează o referință experimentală în memorie volatilă, care oferă o limită superioară de performanță și permite cuantificarea costurilor de transfer. Arhitectura este client-server, cu orchestrare centralizată, distribuție uniformă a instanțelor și execuții sincronizate.

Rețeaua este tratată ca element integrat, cu izolarea traficului de stocare, clase de prioritate și control simetric al fluxului; parametri negociați sunt verificați și corectați pentru a elimina asimetriile. Evaluarea se concentrează pe dimensiuni de bloc, proporții citiri/scrieri, nivel de concurență, politici ale sistemelor de fișiere și organizări pe disc, folosind metrici care surprind media și stabilitatea (predictibilitate, percentile de latență). Graficele evidențiază consistența, filtrează fazele tranzitorii, normalizează intervalele și agregă datele în indicatori sintetici.

Un aport metodologic este reconstrucția matematică a distribuțiilor pe server, plecând de la constrângeri măsurate, pentru a obține coerență cu granularitatea disponibilă pe clienți. Instrumentele sunt minimaliste pentru a nu altera comportamentul. Rezultatele sunt prezentate ca hartă a compromisurilor, permițând identificarea configurațiilor echilibrate și integrarea evaluării performanței în procesul de proiectare, cu decizii fundamentate pe observații obiective și reproductibile.

Obiective urmărite în cadrul lucrării

- **Ob.1.** Realizarea unei analize aprofundate a platformelor open-source pentru managementul infrastructurilor de tip cloud (CMP), cu accent pe soluțiile orientate către furnizarea serviciilor de tip Infrastructure-as-a-Service (IaaS). Această analiză urmărește identificarea caracteristicilor funcționale, arhitecturale și de

interoperabilitate, evidențiind avantajele și limitările fiecărei platforme în raport cu cerințele mediilor educaționale.

- **Ob.2.** Investigarea tehnologiilor și mecanismelor de stocare partajată utilizate în infrastructurile de cloud computing. Scopul este de a clasifica soluțiile existente, de a evidenția modul în care acestea gestionează performanța, scalabilitatea și fiabilitatea și de a identifica implicațiile arhitecturale ale alegerii fiecărei tehnologii.
- **Ob.3.** Elaborarea unei metodologii de caracterizare a tiparelor de activitate de intrare/ieșire atât la nivelul hipervizoarelor, cât și la nivelul subsistemelor de stocare. Metodologia propusă vizează obținerea unui model detaliat al comportamentului sistemelor virtualizate din punct de vedere al subsistemelor de stocare, corelând caracteristicile fluxurilor de date cu elementele arhitecturii de stocare.
- **Ob.4.** Proiectarea și implementarea unei arhitecturi experimentale destinate colectării de măsurători precise pentru determinarea profilurilor de citire și scriere în medii virtualizate. Această arhitectură este construită pentru a oferi vizibilitate completă asupra întregului traseu al datelor și pentru a refolosi datele obținute în vederea identificării performanțelor sistemelor de stocare partajate.
- **Ob.5.** Conceperea și dezvoltarea unor module software specializate pentru colectarea și analiza operațiilor de intrare/ieșire în cadrul arhitecturii experimentale. Aceste module sunt proiectate pentru a asigura prelucrarea eficientă a datelor, furnizarea de metrice detaliate și generarea unor rapoarte și diagrame pentru o analiză care să susțină procesul de evaluare și optimizare.
- **Ob.6.** Definirea unei metodologii multicriteriale de evaluare a sistemelor de stocare partajată în rețele de calculatoare, orientată spre identificarea configurațiilor optime pentru funcționarea sub sarcină. Această abordare urmărește corelarea performanței cu parametrii arhitecturali și operaționali, pentru a sprijini luarea deciziilor tehnice fundamentate pe dovezi experimentale.
- **Ob.7.** Determinarea experimentală a repartițiilor blocurilor pentru operațiile de intrare/ieșire, pentru fiecare direcție (citire/scriere), pentru mai multe scenarii de încărcare. Aceste repartiții trebuie obținute pentru două puncte de monitorizare, la nivelul hipervizoarelor și la nivelul subsistemului de stocare pentru a putea fi folosite ulterior în cadrul testelor experimentale pe o infrastructură reală.
- **Ob.8.** Validarea metodologiei propuse prin desfășurarea de campanii experimentale pe infrastructuri reale de stocare. Testele realizate permit verificarea relevanței rezultatelor, demonstrând aplicabilitatea metodei în scenarii complexe și variate.

Conținutul capitolelor

Capitolul I explică fundamentele cloud computing și ale infrastructurilor virtualizate, de la livrarea resurselor ca servicii cu alocare dinamică, scalabilitate și izolare, până la rolul critic al stocării. Sunt descrise accesul pe blocuri și pe fișiere, impactul dimensiunii blocurilor și al adâncimii cozilor asupra debitului, latenței și rezilienței, nivelurile RAID și tehnologiile de rețea pentru stocare cu latență redusă, precum și aspecte de securitate pentru stocarea partajată.

Capitolul II analizează comparativ platformele CMP open source OpenStack, CloudStack și OpenNebula. Pentru fiecare sunt prezentate componentele, serviciile și instrumentele de administrare, apoi urmează o comparație pe arhitectură, rețea, stocare, hipervizoare, formate de imagini și integrare. Concluzia justifică alegerea OpenNebula pentru arhitectura experimentală, datorită echilibrului între simplitate, flexibilitate și integrarea ușoară a extensiilor necesare.

Capitolul III motivează nevoia unei arhitecturi experimentale care să surprindă tiparele reale de acces la stocare și definește un cadru de măsurare pe două niveluri, la hipervizoare și la serverul de stocare. Sunt descrise modulele software, infrastructura hardware, optimizările de sistem și rețea și utilizarea unui spațiu de lucru în memoria volatilă pentru a elimina limitările mediilor persistente. Se precizează criteriile de alegere a platformei de virtualizare și parametrii planificatorului.

Capitolul IV prezintă metodologia experimentală de evaluare a performanței. Sunt definite obiectivele, fluxul rulărilor, mărimile măsurabile și diagramele generate. Profilurile se

construiesc la nivelul hipervizoarelor, apoi se deduc repartițiile blocurilor la nivelul serverului folosind două modele: o aproximare liniară între distribuții de referință și o metodă bazată pe minimizarea entropiei relative. Capitolul include și o procedură de evaluare multicriterială cu reprezentări grafice comparative.

Capitolul V prezintă rezultatele. Mai întâi se validează infrastructura de monitorizare prin corelarea seriilor temporale între hipervizoare și server, cu analiza diferențelor integrale și instantanee și concluzii privind sincronizarea. Apoi se determină profilurile de intrare și ieșire pentru scenarii de încărcare medie și maximă, cu tabele, repartiții ale dimensiunilor de bloc și rapoarte citire scriere. În final sunt evaluate sisteme de stocare partajată pentru combinații de sisteme de fișiere și niveluri RAID, cu descrierea parametrilor, a comportamentului sub sarcină și a observațiilor extrase din grafice.

Capitolul VI sintetizează obiectivele și etapele cercetării, inventariază contribuțiile originale ale arhitecturii și metodologiilor propuse și conturează direcții de continuare. Ansamblul demonstrează că profilurile reale de utilizare sunt esențiale pentru evaluarea corectă a performanței și predictibilității stocării în medii virtualizate și că integrarea dintre fundamente teoretice, arhitectură experimentală și algoritmi de analiză oferă un cadru practic pentru selecția configurațiilor adecvate.

Capitolul I - Arhitecturi pentru cloud computing și sisteme de stocare partajate - stadiul actual

1.1. Arhitecturi pentru cloud computing

1.1.1. Beneficiile cloud computing-ului în procesele educaționale

Cloud computing-ul este un model de furnizare a resurselor informatice-putere de procesare, stocare, rețea privată sau cu acces public, aplicații software-ca servicii accesibile la distanță, permițând utilizarea fără deținerea sau administrarea directă a echipamentelor [1]. Posibilitatea instanțierii rapide a sistemelor virtualizate și a unui mediu de lucru dedicat constituie un avantaj esențial în infrastructurile moderne [2], iar interesul educațional pentru aceste tehnologii este constant de peste două decenii [3]. Tehnologiile de virtualizare și serviciile cloud permit, în câteva secunde, crearea de mașini virtuale preconfigurate pentru dezvoltare software, procesare de date, simulări științifice sau instruire practică, o opțiune deosebit de atractivă într-un context educațional cu nevoi diverse și neuniforme [4].

Caracteristicile definitorii, serviciu la cerere, elasticitate, acces prin rețea, resurse partajate, servicii măsurabile, generează avantaje semnificative față de infrastructurile clasice [5], [6].

Adoptarea cloud-ului a susținut educația la distanță și tranziția accelerată către învățământul online în perioada pandemiei COVID-19, asigurând continuitatea cursurilor, evaluărilor și colaborării între studenți și cadre didactice [7] și [8]. Exemple concrete includ laboratoare virtuale pentru discipline tehnice: transformarea unui laborator fizic de rețelistică într-unul virtual bazat pe cloud, folosind instrumente precum GNS3 și Cisco Packet Tracer [6]; respectiv implementarea unei arhitecturi VDI (Virtual Desktop Infrastructure) pentru furnizarea de desktop-uri studenților, pe un cloud privat construit cu OpenNebula [9]. Cloud-ul contribuie la democratizarea accesului la tehnologii avansate, făcând posibile proiecte educaționale complexe și de cercetare chiar și în instituții cu resurse limitate [10].

În domeniul CTI, cloud computing-ul permite livrarea rapidă de medii virtuale personalizate: imagini cu baze de date relaționale (MySQL, PostgreSQL, Oracle) pentru laboratoare; scenarii automatizate și medii controlate pentru securitate cibernetică (inclusiv exerciții de tip “capture-the-flag” și răspuns la incidente); topologii de rețea complexe cu rutere, firewall-uri și distribuitoare de încărcare; suport pentru studii avansate de sisteme de operare (nuclee personalizate, planificare, managementul memoriei); clustere virtuale pentru programare paralelă și distribuită (MPI, OpenMP, Hadoop); medii DevOps pentru CI/CD, orchestrare Kubernetes, automatizări cu Ansible/Terraform; acces la toolchains pentru cursuri teoretice (limbaje formale, compilatoare). Pentru inteligență artificială și învățare automată, se pot utiliza imagini preconfigurate cu TensorFlow, PyTorch sau scikit-learn, eventual cu GPU virtualizat, iar în etapele inițiale ale programării se pot furniza medii standardizate cu IDE-uri și compilatoare preinstalate.

Accesul la medii identice, resetabile și independente de platforma locală uniformizează experiența de învățare, reduce efortul personalului tehnic și apropie instruirea de condițiile reale din industrie[9].

Realizarea deplină a acestor beneficii depinde de proiectarea și administrarea unei infrastructuri cloud cu performanță ridicată și disponibilitate constantă. Factorii determinanți includ viteza rețelei, capacitatea și funcționarea optimă a stocării, managementul resurselor de procesare și eficiența mecanismelor de orchestrare, alături de optimizări pentru trafic, latență și scalabilitate elastică, astfel încât variațiile cererii să fie absorbite fără compromiterea stabilității întregului sistem.

1.1.2. Modele de arhitecturi pentru cloud

Cloud computing-ul s-a conturat ca model de livrare a resurselor informatice prin analogie cu utilitățile (electricitate/gaz), idee anticipată încă din anii '70 de Leonard Kleinrock prin conceptul de „computer utilities”, servicii disponibile la cerere [11]. Realizarea acestei viziuni în ultimele două decenii a fost posibilă prin convergența virtualizării, a accesului extins la rețea (Internet), a creșterii capacității centrelor de date și a maturizării tehnologiilor de management automatizat al resurselor [1], [11], [12]. Conceptual, cloud computing-ul livrează servicii IT-de la infrastructură hardware la platforme și aplicații complete-„la cerere”, de regulă cu tarificare pe consum [11]. Arhitectura este stratificată, structurată pe trei modele majore: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) și Software as a Service (SaaS), fiecare oferind niveluri distincte de abstracție și control, de la administrarea hardware-ului la aplicații gata de utilizare [1].

Din perspectiva modului de acces și a proprietății asupra infrastructurii, cloud-ul se clasifică în public, privat și hibrid. Cloud-ul public este deținut și operat de furnizori terți, cu resurse partajate și acces prin Internet, optimizat pentru cost și scalabilitate; cloud-ul privat este dedicat unei singure organizații, cu control sporit al securității și conformității, însă cu costuri/efort operațional mai mari [13]; cloud-ul hibrid combină cele două modele, permițând migrarea flexibilă a sarcinilor pentru a echilibra cost, performanță și securitate [14].

Complexitatea domeniului a generat propuneri de taxonomii care să sistematizeze arhitecturile cloud: componente de infrastructură, modele de servicii (IaaS, PaaS, SaaS) [15], modele de implementare (public/privat/hibrid/comunitar), mecanisme de virtualizare, instrumente de orchestrare și automatizare, precum și tehnologii emergente-container, microservicii, funcții serverless [16]. O astfel de taxonomie susține atât înțelegerea teoretică, cât și proiectarea și optimizarea soluțiilor la scară. De asemenea, analiza controlului entităților și a responsabilităților pe straturi completează cadrul conceptual [17], [18].

În practică, organizațiile adoptă pe scară largă arhitecturi hibride și strategii multi-cloud pentru a evita blocarea la furnizor și pentru a optimiza costurile/performanțele. În paralel, paradigmele serverless și Function as a Service (FaaS) câștigă teren: funcții scurte, declanșate de evenimente, executate fără gestionarea explicită a serverelor, cu facturare pe timpul efectiv de execuție. Tendințele curente includ integrarea inteligenței artificiale și a învățării automate direct în arhitecturi, optimizarea dinamică a costurilor prin orchestrare inteligentă și extinderea edge computing-ului pentru reducerea latenței-indicând rolul strategic al cloud-ului într-o economie digitală competitivă [8].

În consecință, selecția între IaaS, PaaS și SaaS trebuie să echilibreze controlul tehnic, flexibilitatea, costurile și riscurile, în acord cu capacitatea organizației de a răspunde rapid schimbărilor tehnologice și de business [19]. În mediul educațional, adoptarea cloud-ului facilitează colaborarea și accesul la resurse/laboratoare/aplicații din orice locație, sprijinind învățarea la distanță și flexibilitatea operațională [20]; platformele open-source pot fi personalizate/extinse în funcție de nevoi instituționale [21]. Totodată, performanța depinde de configurația hardware, parametrii de rețea și optimizarea stocării, necesitând testare inițială și evaluare continuă pentru o experiență optimă a utilizatorilor [22].

1.1.2.1. IaaS - Infrastructure-as-a-Service

Infrastructure-as-a-Service (IaaS) este stratul de bază al cloud computing-ului și furnizează ca servicii resursele fundamentale de calcul (procesoare, memorie RAM, stocare, conectivitate de rețea, GPU), accesibile prin API-uri (în linie de comandă sau REST) ori interfețe web [12]. Esența IaaS constă în abstractizarea completă a infrastructurii fizice, astfel încât utilizatorii își pot configura și administra propriile sisteme de operare, echipamente virtuale de rețea, aplicații și date, fără a se preocupa de detalii legate de hardware sau de infrastructura fizică [21]. Modelul răspunde nevoii organizațiilor de a elimina investițiile mari în servere și infrastructură IT, oferind închirierea flexibilă a resurselor în funcție de necesități și posibilitatea de a scala rapid capacitatea, cu plată doar pentru resursele efectiv consumate și/sau utilizate.

IaaS conferă un grad ridicat de control utilizatorului: spre deosebire de PaaS sau SaaS, accesul direct la resursele virtuale permite instalarea sistemelor de operare, definirea rețelelor

virtuale și implementarea unor politici complexe de securitate și izolare [12]. Arhitectural, hipervizorul asigură rularea simultană, cu izolare, a mai multor mașini virtuale pe același hardware [22], iar deasupra nivelului de virtualizare intervin module de management, orchestrare și alocare a resurselor [23]. Sistemul de stocare este expus sub forma unor volume virtuale atașabile/detașabile, cu posibilitatea de snapshot, replicare și migrare flexibilă a încărcării fără oprirea mașinilor virtuale. Virtualizarea rețelei permite definirea topologiilor proprii, a firewall-urilor, VPN-urilor și a rețelelor virtuale complet izolate, chiar dacă traficul fizic traversează aceleași echipamente.

Adopția pe scară largă este motivată de beneficii precum flexibilitatea și scalarea rapidă a infrastructurii pentru variații sezoniere ale cererii, situații neprevăzute sau evenimente majore [11]. Pe de altă parte, gestionarea instanțelor virtualizate este complexă: configurarea și administrarea sistemului de operare, a aplicațiilor și a securității revin utilizatorului, necesitând competențe tehnice, monitorizare, copii de siguranță și actualizări continue. În medii partajate, riscurile de securitate sunt amplificate; deși izolarea este robustă, atacuri de tip side-channel sau vulnerabilități ale hipervizorului pot permite, în teorie, scurgeri de informații, aspecte ce trebuie monitorizate atent, mai ales pentru date sensibile [15].

Pe lângă platformele comerciale, soluțiile open-source includ OpenStack (platformă IaaS completă, utilizată în cloud-uri private și publice), cu componente pentru procesare, rețea, stocare și gestionarea utilizatorilor [12], [22]; CloudStack, apreciată pentru simplitate și stabilitate; și OpenNebula, eficientă în cloud-uri private mici și medii, cu instalare/configurare mai facilă, dar cu funcționalități mai restrânse și scalare mai dificilă în scenarii complexe, conform studiilor comparative [24]. Eucalyptus a fost importantă prin compatibilitatea cu API-urile AWS; în testele de performanță, OpenStack depășește constant CloudStack și OpenNebula la sarcini mari, cu un nivel operațional mai complex [21]. În educație, IaaS susține laboratoare virtuale pentru sisteme de operare și rețele, cu instanțe Linux izolate per student [6], [19]. Direcțiile actuale includ orientarea către DevOps și Infrastructure-as-Code (Terraform, Ansible, Heat) [25], precum și integrarea orchestratoarelor de containere (de exemplu, Kubernetes) pentru densități mai mari și timpi de pornire reduși [26].

1.1.2.2. PaaS - Platform-as-a-Service

Platform-as-a-Service (PaaS) este nivelul intermediar al modelului cloud computing, poziționat între IaaS, care furnizează resurse hardware virtualizate, și SaaS, care oferă aplicații gata de utilizare. PaaS pune la dispoziția dezvoltatorilor un set complet de servicii și instrumente pentru dezvoltare, testare, implementare și scalare, fără administrarea directă a infrastructurii care susține aceste servicii [1], [15]. Paradigma a apărut pentru a accelera ciclul de dezvoltare software și pentru a reduce sarcinile legate de configurarea mediilor de execuție și gestionarea serverelor. Spre deosebire de IaaS, unde clientul gestionează întreaga structură software, în PaaS furnizorul asigură un mediu de execuție pregătit, care poate include servere de aplicații, baze de date, instrumente DevOps și servicii de integrare, iar utilizatorii se concentrează pe logica și dezvoltarea aplicațiilor [2].

Arhitectura PaaS este stratificată. La bază, folosește infrastructură virtualizată, de regulă furnizată prin IaaS, pentru a crea medii de execuție izolate. Deasupra se află un strat de mijloc, care poate include servere web, servere de aplicații, baze de date, servicii de cache, sisteme de mesagerie, soluții pentru autentificare și managementul utilizatorilor. Un element definitoriu este suportul pentru mai multe limbaje de programare și medii de execuție. Platforme precum Google App Engine, Red Hat OpenShift și Heroku acceptă limbaje precum Python, Java, Node.js, Ruby și PHP, iar Red Hat OpenShift integrează containere Docker și folosește Kubernetes pentru izolare și scalare automată. Integrarea serviciilor de baze de date și de stocare este centrală, majoritatea platformelor oferind baze de date gestionate, relaționale sau NoSQL, împreună cu servicii de cache precum Redis sau Memcached, ceea ce transferă către furnizor activități precum configurarea, actualizările de securitate și copiile de siguranță.

Platformele moderne includ instrumente DevOps, de exemplu soluții de Continuous Integration și Continuous Deployment, precum Jenkins și GitLab CI, precum și funcționalități pentru monitorizare și gestionarea fișierelor de jurnal. Există totuși dezavantaje, în special dependența de furnizor, deoarece aplicațiile utilizează adesea API-uri și servicii proprietare,

ceea ce îngreunează migrarea. Controlul limitat asupra infrastructurii poate crea dificultăți când sunt necesare setări speciale de securitate, performanță sau conformitate, aspecte importante în mediile enterprise.

O tendință actuală este includerea conceptului de serverless computing în ecosistemul PaaS. Dezvoltatorii scriu funcții care se execută la evenimente, platforma gestionând orchestrarea și execuția. Servicii precum AWS Lambda, Google Cloud Functions și Azure Functions sunt tot mai utilizate, mai ales în arhitecturi cu microservicii sau IoT, dar pot apărea provocări legate de latența la inițializare și de dependența de furnizor. Utilizarea containerelor este o evoluție semnificativă, acestea partajează același nucleu al sistemului de operare, izolează procese și resurse, reduc supraîncărcarea și timpii de pornire, însă în mediile cu mulți utilizatori persistă preocupări de securitate, deoarece izolarea la nivel de nucleu nu este la fel de robustă ca în cazul hipervizoarelor [11].

1.1.2.3. SaaS - Software-as-a-Service

Software-as-a-Service este componenta din cloud computing care livrează direct utilizatorilor finali aplicații găzduite în infrastructura furnizorilor și accesibile prin Internet, de cele mai multe ori prin intermediul unui navigator web sau prin aplicații dedicate. În acest model, software-ul nu se mai instalează local, ci rulează pe servere centralizate administrate de furnizorii SaaS, care se ocupă de întreținere, actualizări și securitate. Abordarea a schimbat modul de acces la tehnologie, reducând nevoia de infrastructură proprie și costurile de operare, precum și investițiile inițiale [2], [27].

Din punct de vedere tehnologic, SaaS se bazează în mod obișnuit pe arhitecturi partajate, în care aceeași instanță software deservește simultan mai mulți clienți, fiecare având acces exclusiv la propriile date și configurații. În același timp apar provocări legate de securitate și de izolare a datelor între clienți, mai ales în mediile comerciale cu cerințe stricte de confidențialitate și conformitate legală [28]. Soluțiile SaaS moderne integrează mecanisme avansate de securitate, de exemplu criptarea datelor la stocare și în tranzit către clienți, autentificarea multi-factor, audituri de securitate și politici granulare de acces. Furnizori importanți, precum Microsoft cu platforma Office 365, Google Workspace sau Salesforce, implementează standarde riguroase de conformitate, de tip ISO 27001 sau GDPR, pentru a răspunde cerințelor clienților din industrii reglementate [27], [29].

Un aspect distinctiv al SaaS este disponibilitatea aproape instantanee a noilor funcționalități, actualizările fiind livrate centralizat de furnizor și devenind imediat accesibile, fără intervenții manuale sau perioade planificate de întrerupere. Furnizorii adaugă funcționalități inteligente, de exemplu analiză predictivă, automatizări bazate pe învățare automată și asistenți virtuali pentru sprijinirea sarcinilor zilnice. În paralel apar aplicații SaaS dedicate gestionării ecosistemelor IoT sau utilizării blockchain pentru trasabilitatea datelor și securizarea tranzacțiilor. Aceste inovații aduc provocări suplimentare, în special în zona securității și a reglementărilor legale, care vor continua să influențeze ritmul adopției [30], [31].

1.2. Sisteme de stocare partajate pentru cloud computing

Cloud computing-ul a impus o redefinire a conceptelor de stocare, deoarece cerințele actuale de performanță, elasticitate și predictibilitate au depășit rapid limitele soluțiilor tradiționale. Extinderea utilizării infrastructurilor virtualizate și containerizate, împreună cu creșterea volumelor de date generate de aplicațiile distribuite, a accentuat constrângerile privind latența și debitul operațiilor de intrare ieșire. Diversitatea sarcinilor de lucru din cloud, de la baze de date tranzacționale și aplicații analitice complexe, până la big data, inteligență artificială și fluxuri în timp real, impune infrastructuri de stocare capabile să ajusteze rapid lățimea de bandă, numărul de operații pe secundă și latența, în funcție de dinamica cererii.

Aceste cerințe au condus la dezvoltarea mai multor soluții arhitecturale, fiecare cu avantaje și compromisuri specifice. Sistemele de fișiere distribuite, volumele pe blocuri virtualizate și soluțiile hibride se află în centrul strategiilor moderne, oferind flexibilitatea necesară pentru a răspunde nevoilor operaționale. Integrarea acestor tehnologii cu motoarele de orchestrare și cu serviciile de management al resurselor din platformele de cloud contribuie la optimizarea utilizării infrastructurii și la creșterea nivelului de automatizare. Densitatea

sporită a sarcinilor în medii multi-utilizator, extinderea masivă a volumelor de date și evoluția serviciilor care cer procesare paralelă și răspunsuri rapide obligă la soluții de stocare ce livrează debit ridicat și latențe reduse, cu fiabilitate și consistență solide. În consecință, stocarea trebuie tratată integrat cu rețeaua și cu platformele de orchestrare, pentru alocare dinamică și eficiență a resurselor.

Analiza arhitecturilor de stocare, a tipologiilor de infrastructură și a tehnologiilor moderne, precum și înțelegerea implicațiilor asupra platformelor de virtualizare, reprezintă premise pentru proiectarea, optimizarea și implementarea unor soluții capabile să livreze performanța și eficiența dorite [32].

În mediile cloud predomină două paradigme arhitecturale pentru stocare, stocarea la nivel de bloc și stocarea la nivel de fișiere. Alegerea și utilizarea uneia dintre aceste abordări influențează latențele, volumul operațiilor pe secundă, reziliența la incidente critice și complexitatea operațională a sistemului. Decizia trebuie corelată cu profilul de sarcină și cu obiectivele de performanță și disponibilitate, în contextul constrângerilor operaționale specifice platformelor cloud [2], [33], [34].

1.2.1. Stocarea la nivel de blocuri

Stocarea la nivel de blocuri este o abordare fundamentală în care spațiul unui dispozitiv este împărțit în secțiuni uniforme, denumite blocuri binare, care funcționează ca unități elementare pentru operațiile de intrare și ieșire. Dimensiunea blocului este fixă și rezultă din caracteristicile mediului hardware și din standardele tehnice adoptate. În practică, se folosesc în special trei opțiuni, 512 octeți, 4 KB și blocuri cu dimensiune configurabilă. Dimensiunea de 512 octeți provine din standardele mai vechi ale discurilor magnetice și rămâne prezentă pentru compatibilitate. Dimensiunea de 4 KB este tot mai răspândită, inclusiv deoarece, în multe SSD-uri, corespunde paginii interne de memorie pentru scriere, ceea ce influențează modul de procesare a scrierilor și ștergerilor. Tehnologiile moderne, de tip NVMe, oferă blocuri configurabile la nivel de namespace, adică posibilitatea ajustării dimensiunii pentru fiecare spațiu virtual definit, fără a modifica structura fizică internă.

Accesul la date se realizează prin Logical Block Addressing, fiecare bloc fiind identificat numeric, astfel încât software-ul vede un șir ordonat de locații. Acest mecanism a înlocuit complet schemele bazate pe cilindri, capete și sectoare, eliminând limitările arhitecturilor mecanice. Stocarea la nivel de bloc permite operații uzuale, citire, scriere, comenzi Trim, Unmap sau Discard, care anunță dispozitivului că anumite blocuri pot fi eliberate, și mecanisme Flush și Barrier, care obligă golirea cache-ului volatil pe suportul fizic și mențin ordinea corectă a scrierilor, aspect important pentru integritatea datelor.

Nativ, această formă de stocare nu oferă organizare logică în fișiere, directoare sau ierarhii, ci expune o suprafață binară continuă, segmentată în blocuri de dimensiune uniformă. Pentru utilizare practică este necesar un strat software care să transforme blocurile în structuri logice accesibile, în absența lui stocarea rămâne o colecție de date binare cu utilitate directă limitată. Un aspect relevant apare când dimensiunea scrierilor nu coincide cu mărimea blocului, de exemplu o scriere de 1 KB într-un bloc de 4 KB, caz în care dispozitivul nu poate modifica doar porțiunea vizată, ceea ce implică citirea paginii, modificarea în memoria tampon și rescrierea întregului bloc, cu impact asupra performanței, mai ales la SSD-uri.

La un nivel superior dispozitivului, block storage reprezintă o abstracție logică a unei suprafețe continue de blocuri uniforme, distinctă de organizarea fizică pe sectoare. Această separare decuplează granularitatea logică expusă software-ului de granularitatea fizică a mediului, blocul din block storage fiind o unitate de expunere și control. Trecerea de la sectoarele fizice la dimensiunile blocurilor logice mută accentul de pe constrângerile dispozitivului pe definirea unei unități logice de lucru, care stabilește modul în care spațiul este văzut și administrat, independent de suportul fizic.

1.2.1.1. Dimensiunea blocurilor în sisteme de stocare

Dimensiunea blocului este un parametru esențial în sistemele de stocare bazate pe blocuri, cu impact direct asupra performanței, eficienței și durabilității mediilor de stocare [35].

În funcție de tipul aplicațiilor și de modul de acces la date, în practică se disting două categorii principale, blocuri mici și blocuri mari, fiecare cu avantaje și limitări specifice.

Blocurile mici, de exemplu 512 B sau 4 KB, sunt recomandate în sarcini cu accesări aleatorii, când operațiile de citire și scriere vizează porțiuni reduse de date distribuite neregulat. Ele reduc risipa de spațiu, deoarece dimensiunile mici limitează fragmentarea internă, proprietate utilă în medii cu numeroase fișiere de dimensiuni mici. În contrapartidă, pentru a transfera volume mari de date, blocurile mici determină un număr mai mare de operații de intrare/ieșire, ceea ce poate crește încărcarea asupra subsistemelor. Totodată, ele implică un management intern suplimentar, sistemul de stocare trebuind să urmărească un număr semnificativ mai mare de blocuri, fiecare înregistrat în structuri de metadate care reflectă starea și asocierea sa.

Blocurile mari, de exemplu 128 KB sau 256 KB, sunt avantajoase în accesul secvențial, unde transferurile continue ale unor volume mari de date reduc semnificativ numărul de operații de intrare/ieșire. În aceste situații, eficiența hardware crește, fiind necesară mai puțină procesare per operație, ceea ce se traduce printr-un debit mai ridicat și o încărcare redusă asupra controlerelor și magistralelor. Pe de altă parte, în spații cu multe fișiere mici apare riscul de fragmentare internă, deoarece blocurile pot fi ocupate parțial. De asemenea, în accesul aleatoriu pe porțiuni mici, blocurile mari obligă transferul unor cantități mai mari de date decât este necesar, cu impact negativ asupra performanței.

Alegerea dimensiunii blocurilor nu se realizează arbitrar, ci pe baza unor teste de performanță reprezentative pentru sarcina reală. În practică, se variază dimensiunea blocului, de exemplu 4 KB, 16 KB, 64 KB, iar sistemul este evaluat prin indicatori precum numărul de operații pe secundă, lățimea de bandă, latențele medii și maxime și eficiența utilizării spațiului de stocare. O astfel de abordare permite identificarea configurației care echilibrează cerințele aplicațiilor cu constrângerile infrastructurii și caracteristicile mediilor de stocare.

1.2.1.2. Adâncimea coziilor de comenzi

Adâncimea coziilor, I/O queue depth, este un parametru care influențează direct performanța stocării la nivel de bloc, reprezentând numărul maxim de comenzi de intrare ieșire active și în așteptare pe un dispozitiv de stocare [36]. În arhitecturi tradiționale, de tip SATA sau SAS, controlerul gestionează un paralelism limitat, de regulă în zona unităților sau a zecilor de comenzi. Standardul NVMe a schimbat radical acest model, permițând până la 64k de cozi independente, fiecare putând susține până la 64k de comenzi active, concepție adaptată procesoarelor cu multe nuclee și scenariilor în care paralelismul I/O este esențial [37].

Adâncimea coziilor determină cât de mult paralelism poate exploata dispozitivul. O adâncime mai mare permite primirea simultană a mai multor cereri și reordonarea lor pentru optimizare internă, ceea ce se traduce prin rată de transfer mai ridicată, mai ales în acces secvențial, precum și prin utilizarea completă a resurselor hardware, inclusiv magistralele PCI Express și controlerul interne. Totuși, creșterea adâncimii nu este întotdeauna benefică. În scenarii cu acces aleatoriu pe porțiuni mici de date, o adâncime redusă, de exemplu 1 până la 8 comenzi active, poate scădea latența medie, controlerul răspunzând mai rapid fiecărei cereri.

În schimb, adâncimi mari, de ordinul sutelor sau miilor de comenzi active, sunt avantajoase în transferuri masive sau în acces secvențial la date. În aceste situații, paralelizarea amplă maximizează viteza de transfer, exploatează capacitatea internă a dispozitivului NVMe, reduce perioadele de inactivitate ale magistralelor și crește eficiența subsistemului de stocare la nivel de ansamblu. Alegerea adâncimii optime depinde însă de aplicație. Un număr maxim de comenzi simultane nu implică automat performanțe superioare, motiv pentru care testele practice sunt indispensabile.

Evaluarea se face prin simulări cu valori diferite ale adâncimii cozii și măsurarea indicatorilor relevanți, număr de IOPS, lățime de bandă, latențe la citiri și la scrieri. Combinațiile dintre dimensiunile blocurilor și adâncimea coziilor trebuie calibrate atent în funcție de profilurile reale de utilizare și de comportamentul aplicațiilor, astfel încât configurația aleasă să echilibreze debitul, latențele și gradul de paralelism în concordanță cu obiectivele de performanță [38].

1.2.1.3. RAID și protecția datelor

RAID, Redundant Array of Independent Disks, este o tehnologie centrală pentru stocarea la nivel de bloc, proiectată să combine mai multe discuri fizice într-un volum logic unic, cu obiectivul dublu de performanță superioară și protecție împotriva pierderii de date. Aplicațiile interacționează cu volumul rezultat ca și cum ar fi un singur disc cu capacitate mai mare, complexitatea internă fiind ascunsă utilizatorului [39]. RAID operează strict la nivel de bloc, independent de sistemele de fișiere, tratează datele ca o succesiune de blocuri binare și permite paralelizarea scrierilor pe mai multe discuri, ceea ce susține atât creșterea performanței, cât și refacerea datelor în caz de defect. Tehnologia este utilizată de mult timp, în variante hardware și software, iar multiplele configurații disponibile sunt alese în funcție de prioritățile mediului de stocare [40].

Dintre nivelurile uzuale, RAID 0, striping, distribuie blocurile pe toate discurile și oferă performanța cea mai ridicată, însă nu asigură nicio protecție, pierderea unui disc înseamnă pierderea întregului volum. RAID 1, mirroring, duplică identic datele pe două sau mai multe discuri, oferă protecție foarte bună, dar capacitatea utilizabilă se reduce la cea a unui singur disc. RAID 5 combină striping cu paritate distribuită, permite refacerea datelor la defectarea unui singur disc și oferă un compromis între performanță, capacitate și siguranță, scrierile fiind mai lente decât citirile, iar reconstrucția volumelor mari poate fi lentă. RAID 6 extinde modelul cu dublă paritate, rezistă la defectarea simultană a două discuri, scrierile sunt și mai costisitoare computațional, iar capacitatea utilizabilă scade suplimentar, echivalent cu dimensiunea a două discuri [40].

În infrastructurile moderne se folosesc frecvent niveluri combinate, pentru echilibru optim între performanță, protecție și utilizarea spațiului. RAID 10, 1 plus 0, combină mirroring și striping, oferă performanță foarte mare și siguranță bună, cu înjumătățirea capacității utilizabile. RAID 50 creează grupuri RAID 5 agregate prin striping, crește performanța față de RAID 5 simplu și tolerează defectarea unui disc în fiecare grup. RAID 60 extinde RAID 6 cu striping între grupuri, furnizează dublă paritate pe grup și performanță crescută, fiind potrivit pentru medii care cer integritate ridicată a datelor [40], [41].

Alegerea nivelului RAID nu se rezumă la calcule de capacitate sau la valori brute de performanță, ci presupune evaluarea modelelor de acces la date, a cerințelor aplicațiilor și a riscului acceptat pentru pierderi de informație ori perioade de indisponibilitate. În această perspectivă, RAID este o componentă frecvent integrată în arhitecturile de stocare, cu rol concomitent de protecție și de susținere a performanței.

1.2.2. Stocarea bazată pe fișiere

Stocarea la nivel de fișiere organizează datele în mod logic ca fișiere și directoare, gestionate de un sistem de fișiere. Spre deosebire de stocarea bazată pe blocuri, unde aplicațiile operează cu blocuri anonime, aici accesul are loc prin obiecte denumite, fișiere, și prin structuri ierarhice, directoare, fapt care schimbă modul de interacțiune al utilizatorilor și aplicațiilor cu datele [40], [42]. Sistemul de fișiere acționează ca intermediar între procese și suportul fizic, traduce operațiile de nivel înalt, deschidere, citire, scriere, ștergere, în instrucțiuni precise pentru manipularea blocurilor de pe disc.

Rolurile principale ale unui sistem de fișiere includ organizarea datelor în ierarhii ușor de administrat, gestionarea metadatelor, de exemplu timpi de acces și modificare, drepturi de proprietate, atribute extinse, asigurarea integrității datelor prin mecanisme precum jurnalizarea sau sumele de control și optimizarea performanței prin algoritmi de alocare a blocurilor și politici de caching. Un sistem de fișiere este mai mult decât un container, este alcătuit din structuri interne care gestionează atât datele, cât și metadatele. Printre acestea se numără superblocul, plasat într-o locație bine definită și conținând parametrii globali ai volumului, a cărui corupere poate face volumul inaccesibil, tabelul de inode-uri, câte un inode pentru fiecare fișier sau director, cu metadate precum dimensiune, adresele blocurilor, timpi și drepturi, separarea numelui de metadate permițând căutări eficiente, structurile pentru managementul spațiului liber, hărți de biți sau arbori care urmăresc blocurile libere sau ocupate, și jurnalul, care înregistrează tranzacțional modificările pentru a permite refacerea integrității în caz de erori [44].

De-a lungul deceniilor, sistemele de fișiere au fost element central în gestionarea datelor pe stații de lucru, calculatoare personale și servere. Fiecare implementare reflectă evoluția tehnologică a platformei țintă și cerințele aplicațiilor, diferențiindu-se prin organizarea datelor, strategiile de integritate, metodele de alocare a spațiului, suportul pentru jurnalizare și capacitatea de a gestiona fișiere sau volume mari. Diversitatea implementărilor, ext4, XFS, ZFS, NTFS, FAT32 și exFAT, APFS, HFS+, ReiserFS, F2FS, ilustrează adaptarea la arhitecturi hardware variate și la schimbările rapide din stocare. În continuare sunt prezentate doar câteva dintre principalele sisteme de fișiere utilizate în soluțiile de stocare ale platformelor de virtualizare, selecția vizând relevanța pentru aceste medii.

1.2.2.1. ext4 - Fourth Extended File System

Sistemul de fișiere ext4, Fourth Extended File System, este o evoluție a ext2 și ext3, concepută pentru a răspunde cerințelor de scalabilitate, performanță și reducere a fragmentării, păstrând stabilitatea și compatibilitatea cu predecesoarele sale [43], [44]. În ext2 și ext3, adresarea blocurilor prin pointeri direcți și indirecti introducea niveluri de indirectare pentru fișiere mari și favoriza fragmentarea. Ext4 înlocuiește această mapare cu extents, structuri care descriu intervale contigue de blocuri, reduc cantitatea de metadate și îmbunătățesc performanța la citiri secvențiale, deoarece sunt necesare mai puține accesări de metadate [45]. Un inode poate stoca până la patru extents direct, iar pentru fișiere foarte mari extents suplimentare sunt organizate într-un arbore, B-tree, ceea ce menține eficiența accesului [46].

Ext4 extinde limitele maxime, suportă volume până la 1 EB și fișiere până la 16 TB, prin adresare pe 48 de biți și suport pentru blocuri până la 64 KB. Mecanismul flex_bg, Flexible Block Groups, permite tratarea mai multor grupe de blocuri ca un grup logic mai mare, ceea ce îmbunătățește alocarea de inode-uri și blocuri și reduce fragmentarea, mai ales pentru fișiere mari [47]. „Delayed allocation” amână alocarea blocurilor fizice, păstrând datele în cache până la momentul potrivit, fapt care favorizează alocările contigue, reduce fragmentarea și optimizează traficul de intrare/ieșire. „Multiblock allocation” permite alocarea simultană a mai multor blocuri, optimizând plasarea datelor și reducând accesările bitmap-urilor interne.

Pentru robustețe, ext4 introduce sume de control în jurnal, ceea ce crește protecția integrității metadatelor și datelor în fața opririlor bruște sau blocajelor. Sistemul oferă trei moduri de jurnalizare, „journal”, „ordered”, implicit în multe distribuții, și „writeback”, fiecare cu un compromis specific între consistență și performanță [43]. Dimensiunea inode-ului crește la 256 de octeți, permițând attribute extinse, xattrs, liste de control al accesului, ACL, timpi de înregistrare cu rezoluție mai fină și stocarea directă a descriptorilor de extents.

Ext4 utilizează algoritmi pentru plasarea eficientă a fișierelor, de exemplu grupuri de blocuri neocupate și strategii pentru alocări frecvente, ceea ce susține performanța. Pentru directoare foarte mari, ext4 înlocuiește listele secvențiale din ext2 și ext3 cu HTree, o variantă optimizată de B-Tree, care accelerează căutarea, inserarea și ștergerea intrărilor. Suportul pentru Discard, Trim, informează dispozitivele de stocare că anumite blocuri pot fi eliberate, aspect important pentru SSD-uri. Din perspectiva securității, ext4 integrează criptarea la nivel de sistem de fișiere prin fscrypt, permite criptarea fișierelor sau directoarelor cu chei gestionate în nucleu, cu transparență pentru aplicații. În plus, Project Quotas adaugă posibilitatea limitării spațiului la nivel de director și subdirectoare, utilă în medii multiutilizator și pentru servicii de găzduire [47].

1.2.2.2. XFS

XFS este un sistem de fișiere inițial dezvoltat de SGI și introdus în 1994 pe IRIX, ulterior portat pe Linux în 2001, devenind o referință pentru volume mari și aplicații cu cerințe ridicate de performanță [48]. Conceput de la început pentru paralelism și scalabilitate, XFS se distinge de ext2 sau ext3 prin utilizarea extensivă a arborilor B+ pentru aproape toate structurile interne, ceea ce oferă operații cu complexitate $O(\log n)$ și înlocuiește listele secvențiale din sistemele mai vechi [49].

Alocarea spațiului este realizată prin extents, care descriu intervale contigue de blocuri și susțin eficient fișiere foarte mari. Inode-urile au o organizare flexibilă, adaptată obiectului stocat, putând include direct extents, attribute extinse și liste de control al accesului, ceea ce

reduce accesările suplimentare și accelerează operațiile. XFS folosește alocare întârziată, datele sunt păstrate temporar în memoria cache, iar atribuirea blocurilor se face ulterior, cu preferință pentru regiuni contigue, fapt care reduce fragmentarea și favorizează scrierile secvențiale.

XFS este proiectat pentru volume foarte mari, limitele teoretice includ sisteme de fișiere și fișiere de până la 8 EB, iar mărimea blocului poate fi configurată la creare între 512 octeți și 64 KB [50]. Mecanismul de alloc groups împarte sistemul de fișiere în grupuri independente cu structuri proprii de alocare, ceea ce permite paralelizarea operațiilor, reduce concurența la blocare pe sisteme multi core și optimizează mișcările capetelor pe discuri mecanice. Integritatea metadatelor este protejată prin jurnalizare tranzacțională de tip write ahead log, cu sume de control, iar sistemul nu jurnalizează datele. Dynamic Inode Allocation permite crearea dinamică a inode-urilor, evitând limitările unei alocări fixe.

Directoarele de dimensiuni foarte mari sunt gestionate tot prin arbori B+, ceea ce susține milioane de intrări cu performanță stabilă, operațiile de căutare, inserare și ștergere având complexitate $O(\log n)$. XFS integrează suport pentru TRIM, informând SSD-urile despre blocurile eliberate, util pentru garbage collection și pentru limitarea amplificării scrierilor. În privința criptării, XFS nu include un mecanism nativ, criptarea se realizează prin soluții externe, de exemplu LUKS, sau la nivelul aplicațiilor [51].

Sistemul oferă management avansat al cotelor, la nivel de utilizator, grup și proiect, proiect quotas, precum și funcționalități precum Online Defragmentation și Online Resizing. Suplimentar, Realtime Subvolumes sunt destinate scenariilor de timp real, unde este necesară o performanță predictibilă a operațiilor de intrare ieșire. În ansamblu, XFS combină structuri de date eficiente, mecanisme de alocare adaptative și suport pentru volume foarte mari, fiind potrivit pentru medii profesionale și de calcul intensiv [48], [49], [50], [51].

1.2.3. Sisteme de fișiere avansate și în rețea

Volumul este o abstracție intermediară între nivelul fizic al dispozitivelor și nivelul logic al sistemelor de fișiere, separare care simplifică administrarea resurselor, permite alocarea eficientă a spațiului și susține ajustarea rapidă a capacității în funcție de cerințele aplicațiilor. Un volum agregă spațiul de pe unul sau mai multe dispozitive fizice și se prezintă sistemului de operare ca un spațiu contiguu și unitar. Spre deosebire de partițiile tradiționale, limitate de dimensiunea și structura unui singur disc, volumele logice se pot extinde pe mai multe discuri, pot fi redimensionate dinamic și pot implementa funcționalități precum duplicarea în timp real, distribuirea pe benzi și imagini instantanee.

Importanța acestei arhitecturi a crescut odată cu apariția virtualizării, containerizării și a infrastructurilor cloud, care impun elasticitate și scalabilitate. În astfel de medii, volumele logice sunt utilizate pentru crearea rapidă a spațiilor de stocare pentru mașini virtuale sau containere, pentru redundanță și performanță prin tehnici software de tip RAID, pentru copii instantanee destinate copiilor de siguranță sau replicării între centre de date și pentru optimizarea utilizării spațiului prin alocare dinamică. Soluții precum Logical Volume Manager, BTRFS și ZFS au evoluat în platforme de management al stocării, integrând funcții avansate fie în sistemele de fișiere, fie în straturi intermediare ale stivei. Deoarece în această lucrare va fi folosit doar ZFS, doar acesta va fi descris în capitolul următor.

O categorie distinctă este cea a sistemelor de fișiere distribuite de tip cluster. Acestea sunt proiectate pentru acces concurent, coerent și cu latență redusă la aceleași date, stocate pe un volum comun partajat fizic. Ele implementează blocare distribuită, jurnalizare sincronizată și gestionare partajată a metadatelor, pentru menținerea consistenței indiferent de numărul de clienți sau de nivelul de încărcare. Exemple reprezentative sunt GFS2, dezvoltat de Red Hat și utilizat în medii critice [52], OCFS2, integrat în nucleul Linux și orientat către infrastructuri enterprise, cu jurnalizare avansată și blocare distribuită [53], GPFS, IBM Spectrum Scale, destinat clusterelor mari și aplicațiilor de performanță ridicată [54].

Separat de acestea există sisteme de stocare distribuite cu scalare orizontală, care nu presupun acces concurent la aceleași blocuri fizice. Datele sunt fragmentate și distribuite între servere, iar un strat software oferă o vedere logică unificată. Reziliența este asigurată prin replicare multiplă sau codare pentru corecția erorilor, iar scalarea se face prin adăugarea de

noduri. Un exemplu este Ceph, bazat pe RADOS, Reliable Autonomic Distributed Object Store, care elimină punctele unice de defectare și oferă scalabilitate liniară, cu CephFS furnizând o interfață compatibilă POSIX peste această arhitectură [55], [56]. Alte implementări, de tip GlusterFS, Lustre sau MooseFS, urmează același model conceptual. În această lucrare, analiza tehnică se va concentra exclusiv asupra OCFS2, ales deoarece a fost folosit în experimente.

1.2.3.1. ZFS

ZFS, conceput ca sistem de fișiere și mecanism integrat de gestionare a volumelor, a fost proiectat de Sun Microsystems și introdus în Solaris în 2005, cu obiective de scalabilitate ridicată, protecție a integrității datelor și administrare eficientă [57]. După deschiderea codului sub numele OpenZFS, tehnologia a depășit cadrul Solaris și este folosită pe Linux și FreeBSD, precum și în soluții comerciale. Fundamentul arhitecturii îl reprezintă „pool”-ul de stocare, Zpool, care agregă dispozitivele fizice într-un spațiu comun din care spațiul este alocat dinamic către datasets sau către volume bazate pe blocuri. Această abordare elimină partiționarea fixă și permite extinderea capacității prin adăugare de discuri fără oprire, optimizând utilizarea și simplificând administrarea [58].

ZFS folosește Copy on Write, orice modificare se scrie în blocuri noi, fapt care asigură consistența metadatelor și protejează datele la întreruperi. Același logică stă la baza copiilor instantanee și a clonelor, utile pentru versiuni, testare și revenire la stări anterioare. Arhitectura internă include Data Management Unit, ZFS Intent Log pentru operații sincrone, ARC, un cache adaptiv în RAM, L2ARC, extensie pe SSD, și VDEV, dispozitive virtuale care compun pool-urile în configurații cu discuri individuale, oglinzi sau RAID Z [58].

Integritatea este verificată prin sume de control aplicate datelor și metadatelor, fiecare bloc are o sumă de control verificată la citire, iar în caz de neconcordanțe se încearcă recuperarea din copii redundante, în funcție de configurația pool-ului [40]. RAID Z elimină vulnerabilitatea „write hole” din RAID 5 sau RAID 6, deoarece datele și paritatea sunt scrise tranzacțional și atomic. Există trei variante, RAID Z1, cu toleranță la defectarea unui disc, RAID Z2, cu toleranță la două discuri, și RAID Z3, cu toleranță la trei discuri.

ZFS oferă opțiuni de configurare pentru performanță, de exemplu ajustarea recordsize la profilul de I/O, alegerea algoritmilor de compresie sau optimizarea structurii VDEV și a jurnalului tranzacțional. Algoritmii de compresie integrați, LZ4, GZIP, ZLE, pot fi aplicați datelor și metadatelor. Deduplicarea identifică și elimină blocurile duplicate din același pool, aducând economii de spațiu în scenarii cu imagini identice sau copii de siguranță succesive [59]. Jurnalul ZIL poate utiliza un dispozitiv separat, SLOG, de regulă un SSD, pentru a accelera scrierile sincrone, datele fiind ulterior transferate asincron în pool-ul principal.

În medii de virtualizare, ZFS facilitează clonarea rapidă a imaginilor de disc și integrarea cu replicarea și copiile instantanee. Limitările includ consumul de memorie RAM asociat ARC, recomandarea generală fiind de cel puțin 1 GB pentru fiecare TB de date stocat, precum și faptul că blocurile eliberate rămân indisponibile cât timp există copii instantanee care le referă, ceea ce impune gestionarea atentă a acestora [60]. În ansamblu, ZFS combină protecția prin sume de control, copii instantanee eficiente, compresie și mecanisme de remediere automată, oferind o soluție robustă pentru administrarea volumelor mari de date.

1.2.3.2. OCFS2

OCFS2 este un sistem de fișiere de tip cluster, conceput pentru acces simultan de pe mai multe noduri la același set de date stocate pe un dispozitiv comun la nivel de blocuri, spre deosebire de sistemele distribuite care fragmentează datele între noduri. El funcționează peste un volum partajat și oferă tuturor instanțelor o vedere unificată a datelor, caracteristică esențială pentru aplicații care solicită coerență strictă și latențe reduse [53]. Dezvoltat inițial pentru Oracle RAC, OCFS2 a evoluat într-un sistem cu utilizare generală, este open-source și este integrat în nucleul Linux începând cu versiunea 2.6.16, integrare care a contribuit la o bună performanță în producție [61].

Arhitectura sa se bazează pe componente interdependente, în special pe un manager distribuit de blocări, DLM, care sincronizează accesul concurent la fișiere și la blocuri din fișiere, pe jurnalizare tranzacțională, necesară recuperării după opriri neprevăzute, și pe un

mecanism de „heartbeat”, folosit pentru detectarea pierderii conectivității unui nod și pentru declanșarea izolării, fencing, astfel încât să fie prevenit accesul neautorizat la datele partajate. OCFS2 garantează consistența strictă POSIX, modificările realizate de un nod devin imediat vizibile pe celelalte, iar blocările pot fi partajate pentru citire, exclusive pentru scriere sau de tip range pentru porțiuni din fișiere, utile atunci când se modifică doar fragmente ale unui fișier mare. Un element definitoriu este Distributed Metadata System, care face metadatele accesibile tuturor nodurilor fără un server central, reducând riscul unui punct unic de defecțiune și susținând scalabilitatea.

Jurnalizarea este inspirată din JBD, fiecare nod deține propriul jurnal, ceea ce permite recuperare locală în caz de incident pe nod, însă presupune sincronizare atentă în scenarii cu scrieri frecvente și concurente. OCFS2 poate gestiona volume de ordinul terabytes și milioane de fișiere, include redimensionare online, alocare de blocuri optimizată pentru cluster, attribute extinse și ACL, precum și gestiunea cotelor. Integrarea cu VFS la nivel de nucleu asigură compatibilitate POSIX, sistemul montat pe mai multe noduri comportându-se similar unui sistem local, însă configurarea corectă a mecanismelor de izolare este critică, întârzierile putând conduce la coruperea datelor [62].

OCFS2 nu este proiectat pentru distribuție geografică, are nevoie de acces rapid și sincron la același stoc bazat pe blocuri și nu oferă nativ erasure coding sau striping distribuit între noduri. Avantajele sale includ acces concurent real la aceleași date, performanță ridicată datorită integrării în nucleu, robustețe pentru consistență și integritate, compatibilitate cu infrastructuri SAN și stocare pe blocuri [53].

1.2.3.3. Sisteme de fișiere în rețea

Sistemele de fișiere în rețea furnizează partajarea datelor între mai multe calculatoare prin expunerea unui spațiu de stocare centralizat, accesibil de la distanță aproape indistinct față de stocarea locală. Conceptul a apărut în LAN-urile timpurii, ca răspuns la nevoia de eliminare a duplicării datelor și de centralizare a administrării resurselor [40]. Arhitectura urmărește ca mai mulți clienți să acceseze simultan aceleași directoare sau același volum, cu o izolare mai bună a datelor și cu reducerea riscului de pierdere ori corupere cauzate de defecte locale de hardware.

Rolul principal este furnizarea unui spațiu de stocare accesibil printr-un protocol standardizat. Operațiile obișnuite asupra fișierelor, deschidere, citire, scriere, ștergere, redenumire, listare de directoare și modificare de attribute, sunt traduse în cereri client-server prin protocoale precum NFS sau SMB, astfel încât utilizatorii și aplicațiile să interacționeze cu resursele de la distanță în mod similar celor locale [40]. Sistemele trebuie să gestioneze cinci cerințe majore, concurența accesului, securitatea, coerența datelor, performanța la încărcări ridicate și compatibilitatea multiplatformă.

Modelul clasic include două componente. Serverul de fișiere expune spațiul de stocare, local sau furnizat de soluții externe precum SAN ori volume virtuale, răspunde cererilor, aplică politici de securitate și poate folosi mecanisme de caching pentru a scurta timpii de răspuns. Clientul inițiază solicitările prin protocolul specific, interpretează răspunsurile și integrează resursele la distanță în sistemul local, astfel încât fișierele să fie utilizate similar celor stocate pe discurile proprii. În jurul acestui nucleu funcțional gravitează componente de sprijin pentru administrare și securitate, de exemplu servicii de nume, Active Directory sau LDAP, care mapează identități și grupe pentru aplicarea permisiunilor, mecanisme de file locking pentru controlul accesului concurent, cache la nivel de client pentru accelerarea acceselor repetate, protocoale de notificare a modificărilor pentru menținerea coerenței și integrare cu audit și jurnalizare pentru trasabilitate.

Provocările majore privesc consistența cache-ului și riscul de blocaje la nivelul blocărilor, deoarece clienții păstrează copii locale ale datelor, iar modificările serverului trebuie propagate rapid pentru a evita lucrul pe versiuni depășite. Interoperabilitatea între platforme eterogene adaugă complexitate, fiecare sistem de operare având extensii și particularități proprii. Diferențele de model de permisiuni, de tratare a caracterelor speciale din numele fișierelor și de codificare a datelor text pot genera comportamente imprevizibile la partajarea între Windows, Linux și macOS. Proiectarea infrastructurii trebuie să ia în considerare nu doar protocolul utilizat, ci și limitele fiecărui sistem participant, capacitățile rețelei și cerințele

aplicațiilor. În ansamblu, sistemele de fișiere în rețea sunt componente indispensabile ale arhitecturilor IT contemporane, iar implementarea lor corectă condiționează compatibilitatea, performanța și securitatea accesului la date.

1.2.3.4. NFS

Network File System, NFS, este unul dintre cele mai utilizate protocoale pentru partajarea fișierelor în rețele, în special în medii Unix și Linux, unde a devenit standard de facto pentru accesul de la distanță la sisteme de fișiere. Conceput de Sun Microsystems în 1984, NFS permite stațiilor de lucru să acceseze fișiere situate pe servere îndepărtate ca și cum ar fi locale, răspunzând nevoilor mediilor distribuite de a partaja eficient resursele de stocare și de a reduce costurile operaționale [63]. Arhitectural, NFS urmează modelul client server, serverul exportă directoare, iar clienții le montează local. Implementările moderne utilizează Remote Procedure Call, RPC, peste TCP, mai robust decât UDP, folosit inițial, care era vulnerabil la pierderi de pachete și la limite de dimensiune a mesajelor.

În Linux, serverul NFS este integrat în nucleu prin modulul nfsd, rularea în kernel space evită tranzițiile costisitoare către user space și permite integrarea strânsă cu Virtual File System, VFS. Evoluția protocolului a trecut de la modele stateless, NFSv2 și NFSv3, unde serverul nu păstra informații de stare între apeluri, la NFSv4, model stateful, cu sesiuni persistente, identificatori pentru blocaje și gestionare îmbunătățită a permisiunilor prin suport ACL. NFSv4 introduce delegarea, serverul poate acorda unui client drepturi temporare exclusive asupra unui fișier sau director, reducând apelurile de rețea pentru acces repetitiv și crescând performanța atunci când un singur client manipulează intensiv datele. Conceptul de pseudo sistem de fișiere oferă un spațiu de nume coerent, evitând fragmentarea logică a spațiilor exportate [64].

Securitatea a evoluat de la mecanisme bazate pe UID și GID către RPCSEC_GSS, cu integrare Kerberos pentru autentificare și criptarea comunicației. Configurarea exporturilor influențează direct stabilitatea și performanța, parametri precum rsize și wsize, modul async versus sync și numărul de fire nfsd active trebuie ajustați în funcție de cerințe. Alegerea async crește viteza scrierilor, dar expune la riscuri în caz de întreruperi, în timp ce sync privilegiază integritatea cu un cost de performanță.

NFS are un rol important în HPC, centre de date și infrastructuri de virtualizare. În HPC este frecvent utilizat pentru partajare între noduri, dar devine limitativ când numărul de operații concurente depășește capacitatea serverului central, motiv pentru care pNFS și RDMA sunt tot mai prezente [65], [66]. În virtualizare, NFS este folosit pentru stocarea imaginilor și a șabloanelor mașinilor virtuale, facilitând migrarea instanțelor, însă accesul simultan de către mai multe hipervizoare poate necesita optimizări atente pentru a preveni blocajele. Operarea curentă se bazează pe monitorizare detaliată, de exemplu latențe ale apelurilor, rata apelurilor RPC eșuate și volume transferate, pentru a detecta congestii și a planifica extinderi de capacitate. Prin flexibilitate și integrare rapidă, NFS rămâne componentă de bază atât în scenarii clasice, cât și în arhitecturi moderne de cloud privat și hibrid.

1.2.4. Rețele pentru sisteme de stocare (SAN)

Rețelele pentru sistemele de stocare, Storage Area Network, SAN, reprezintă o arhitectură dedicată în centrele de date, utilizată acolo unde stocarea la nivel de blocuri trebuie să îndeplinească cerințe ridicate de performanță, disponibilitate și administrare centralizată [67], [68]. În SAN, resursele de stocare nu mai sunt atașate local fiecărui server, ci sunt accesate printr-o rețea specializată, astfel încât mai multe sisteme de calcul pot utiliza aceleași volume, în mod securizat și eficient.

Beneficiile principale includ: abstractizarea mediului de stocare, serverele văd volumele pe blocuri ca pe discuri locale, fără detalii despre localizarea sau organizarea internă; partajarea eficientă a volumelor între mai multe gazde, cu posibilitatea echilibrării încărcării, migrarea dinamică a sarcinilor și utilizarea mai bună a resurselor; scalabilitate ridicată, atât în capacitate cât și în performanță, fără modificări majore ale structurii logice; administrare centralizată, necesară în medii complexe pentru control unitar al politicilor de acces și al resurselor.

Transferul datelor are loc la nivel de blocuri binare, folosind protocoale optimizate pentru viteză și latență scăzută. Protocoalele frecvent întâlnite în SAN sunt: Fibre Channel, iSCSI, FCoE, NVMe over Fabrics. Fiecare are particularități proprii, răspunzând unor cerințe diferite de performanță, cost și complexitate operațională.

În continuare, sunt vizate doar două protocoale, Fibre Channel, considerat standardul „de facto” în industria SAN, și iSCSI, utilizat pentru realizarea testelor experimentale din lucrarea de față.

1.2.4.1. Fibre Channel

Fibre Channel este un standard dominant în rețelele SAN, proiectat pentru transportul datelor la nivel de bloc într-un mod rapid, sigur și cu debit constant. Spre deosebire de rețelele de comunicații generale, Fibre Channel combină protocolul cu o infrastructură hardware dedicată, alcătuită din comutatoare specializate, adaptoare de magistrală pentru gazde și o structură logică de interconectare, fabric, care determină rutarea cadrelor de date [69].

Caracteristicile esențiale includ viteze de transfer foarte mari, valorile curențe ating 32, 64 și 128 Gbps, care asigură flux de date predictibil pentru aplicații critice, latență foarte scăzută, datorată arhitecturii optimizate pentru acces pe blocuri, livrare ordonată a cadrelor, astfel încât datele ajung în aceeași succesiune în care au fost transmise, precum și identificarea unică a dispozitivelor prin World Wide Name, WWN, ceea ce garantează unicitatea elementelor din rețeaua Fibre Channel. Administrarea introduce mecanismul de zoning, prin care se definesc grupuri logice de dispozitive autorizate să comunice între ele, măsură care izolează traficul, crește securitatea și reduce interferențele între aplicații ce partajează aceeași infrastructură fizică.

Domeniile de utilizare tipice vizează centre de date mari și aplicații critice, stocarea bazelor de date cu volum mare de tranzacții, unde viteza și consistența accesului la stocare persistentă sunt imperative, infrastructuri de virtualizare, unde mai multe gazde necesită acces la aceleași volume cu timpi de răspuns minimi, aplicații financiare, unde întârzierile în accesul la date pot produce pierderi.

Limitările principale privesc costurile ridicate, atât pentru echipamentele specializate, cât și pentru formarea personalului necesar implementării și mentenanței, complexitatea administrării, care presupune cunoașterea mecanismelor interne ale fabricii, inclusiv configurarea zonelor și a rutelor logice, lipsa integrării native cu Ethernet, aspect ce poate complica arhitecturile hibride în care convergența rețelelor este o cerință strategică.

În ansamblu, Fibre Channel rămâne o soluție de referință pentru stocare la nivel de bloc în medii profesionale, atunci când sunt necesare performanță ridicată, latență minimă și fiabilitate strictă, cu prețul investițiilor semnificative și al necesității unui nivel avansat de expertiză operațională [69].

1.2.4.2. iSCSI

iSCSI, Internet Small Computer System Interface, este prezentat ca alternativă viabilă la Fibre Channel în arhitecturile moderne de stocare, mai ales acolo unde se urmărește reducerea costurilor fără compromisuri majore de funcționalitate. Spre deosebire de Fibre Channel, care necesită echipamente și interconectări dedicate, iSCSI folosește rețelele IP, încapsulează comenzile SCSI în TCP/IP și transportă datele de tip bloc peste infrastructura Ethernet existentă, fapt care îl face atractiv în organizații cu rețele IP deja performante sau în care investițiile în hardware specializat nu sunt sustenabile [70].

Modelul arhitectural este definit de inițiator, entitatea care solicită accesul la date, de țintă, componenta din sistemul de stocare care expune volumele pe blocuri și aplică autentificarea și drepturile de acces, și de LUN, identificatorul unității logice văzute de inițiator ca disc virtual sau partiție, accesată la nivel de blocuri. Aceste concepte structurează fluxul iSCSI și separă clar rolurile între client și resursa de stocare [71].

Avantajele operaționale derivă din reutilizarea Ethernet, ceea ce reduce costurile, din flexibilitatea ridicată în implementare, iSCSI fiind suportat larg de platforme hardware și software, din posibilitatea extinderii pe distanțe mari, utilă pentru replicare și recuperare în caz de dezastru, și din simplitatea configurării și administrării, instrumentele software fiind integrate

în multe platforme. În contrapondere, sunt menționate limitări care trebuie evaluate înainte de implementare, în special latența mai ridicată față de Fibre Channel, datorată suprasarcinilor TCP/IP, sensibilitatea la congestia rețelei, traficul de stocare împărțind aceeași infrastructură cu traficul de date obișnuit, și dependența de proiectarea atentă a rețelelor în scenarii cu volume mari de date [73].

Din perspectiva utilizării, iSCSI este frecvent implementat în centre de date mici și medii, unde se caută un echilibru între cost și performanță. În medii virtualizate, permite mai multor servere să acceseze același volum pe blocuri, facilitează migrarea dinamică a mașinilor virtuale și susține continuitatea serviciilor în timpul intervențiilor planificate sau neprevăzute. De asemenea, este potrivit pentru soluții de copii de siguranță și replicare, datorită compatibilității bune cu infrastructurile existente și a flexibilității în configurarea legăturilor la distanță.

În concluzie, iSCSI extinde accesul la arhitecturi SAN printr-o abordare pragmatică bazată pe IP, oferă beneficii clare de cost și de integrare și impune, pentru atingerea performanțelor și disponibilității dorite, o evaluare atentă a caracteristicilor rețelei și a profilului aplicațiilor [70], [71], [73].

1.3. Tehnologii de rețea în arhitecturi de stocare performante

Performanța subsistemului de stocare a devenit esențială pentru eficiența operațională a aplicațiilor și serviciilor, pe fondul creșterii exponențiale a volumelor de date și al cerințelor tot mai stricte privind disponibilitatea, latența scăzută și viteza de acces. În acest context, rețeaua a trecut de la rolul de canal de transport la statutul de factor determinant al performanței stocării, deoarece interconectarea eficientă între componentele infrastructurii condiționează direct timpii de răspuns și debitul operațiilor.

Arhitecturile tradiționale, bazate pe TCP/IP și pe paradigma stocării locale, își arată limitele în fața sarcinilor moderne caracterizate prin acces concurent intens, operații aleatorii și transferuri masive. Pentru a depăși aceste constrângeri, au fost adoptate tehnologii orientate către reducerea latenței și descărcarea procesorului: RDMA, RoCE și iWARP, împreună cu standarde de optimizare a rețelei precum Data Center Bridging. Aceste soluții scurtează lanțul de procesare al datelor și diminuează implicarea CPU, permițând serverelor să aloce mai multe cicluri pentru sarcinile de calcul, ceea ce se traduce în beneficii concrete pentru serviciile de stocare distribuită [74].

Integrarea acestor mecanisme în infrastructurile contemporane are consecințe care depășesc simpla creștere a performanței. Sunt afectate deciziile de proiectare arhitecturală (de exemplu alegerea căilor de date și a punctelor de contenție), politicile de securitate (care trebuie să țină cont de noile canale și de modelele de acces direct la memorie) și strategiile de scalare (care devin dependente de predictibilitatea rețelei). Pe măsură ce aplicațiile migrează către cloud și către arhitecturi distribuite, rețeaua capătă rolul unei magistrale de date capabile să transporte volume foarte mari cu predictibilitate și fiabilitate sporite, ceea ce reconfigurează prioritățile în proiectarea subsistemelor de stocare [75].

Secțiunea următoare are ca obiectiv o prezentare succintă a acestor tehnologii, cu accent pe mecanismele interne, avantajele și limitările lor, precum și pe implicațiile practice în sistemele moderne de stocare distribuită. Accentul cade pe analiza relației dintre latență, debit și utilizarea resurselor de calcul, în corelație cu cerințele aplicațiilor, astfel încât concluziile să reflecte constrângerile reale ale unei infrastructuri în care rețeaua este parte integrantă a performanței stocării.

1.3.1. RDMA - Principii și mecanisme

RDMA, Remote Direct Memory Access, este prezentată ca un mecanism cu impact important în arhitecturile distribuite pentru stocare, permițând transferul direct al datelor între regiuni de memorie ale nodurilor fără implicarea procesorului și fără traversarea stivei TCP/IP, cu efecte directe asupra latenței, lățimii de bandă percepute și încărcării CPU [76]. În locul fluxului clasic TCP/IP (copieri între spațiul utilizator și nucleu, segmentare TCP, procesare IP,

reconstrucție la recepție), un adaptor RDMA specializat, RNIC, execută independent copia între memoria sursă și destinație prin mecanisme DMA (Direct Memory Access), după ce aplicația configurează parametrii transferului [77]. RNIC-urile sunt proiectate pentru un număr foarte mare de I/O concurente și latențe foarte mici (de ordinul nanosecundelor până la microsecunde), în funcție de tehnologia de transport utilizată (InfiniBand, RoCE, iWARP) [78].

Canalul logic de comunicație este definit de perechi de cozi (queue pairs, QP), compuse din coada de trimitere (Send Queue) și coada de recepție (Receive Queue) [79]. Înaintea transferurilor, aplicația trebuie să înregistreze explicit regiunile de memorie (pinning), care primesc chei de memorie, lkey pentru acces local și rkey pentru acces de la distanță. Blocarea memoriei influențează arhitectura aplicațiilor: în prezența multor conexiuni, volumul de RAM "pinned" poate deveni considerabil, ceea ce impune politici stricte de reutilizare a buffer-elor. Un avantaj central este modelul zero-copy (datele sunt mutate direct între memoriile aplicațiilor, fără buffer-ele nucleului), care reduce suprasarcina de I/O și consumul de CPU [80]. Finalizarea operațiilor este semnalată prin cozi de finalizare (Completion Queues), pe care aplicațiile le pot interoga sau pot folosi întreruperi, facilitând modele neblocante și lansarea concurentă a multor operații.

Integrarea ridică însă provocări: programatorul preia responsabilități pe care TCP/IP le maschează (crearea și distrugerea QP-urilor, managementul memoriei înregistrate, sincronizarea, tratarea erorilor), iar depanarea este dificilă deoarece instrumentele uzuale de analiză a traficului (de exemplu tcpdump, Wireshark) nu observă fluxurile RDMA, acestea fiind procesate integral în hardware. RDMA dispune de trei familii de transport: InfiniBand (latență minimă și QoS la nivel hardware), RoCE (RDMA peste Ethernet, necesită rețele configurate strict pentru a evita pierderile) și iWARP (RDMA peste TCP/IP, cu garanții de livrare fără schimbări în rețea) [74]. Alegerea depinde de priorități: latență minimă (InfiniBand), reutilizarea infrastructurii Ethernet (RoCE), compatibilitate și simplitate operațională (iWARP).

Exploatarea eficientă necesită reglaje fine ale parametrilor critici (MTU, numărul maxim de QP, capacitatea buffer-elor din cozi, nivelul de agregare a întreruperilor). Configurarea neadecvată poate duce la subutilizare, pierderi de pachete sau blocaje, iar limitele hardware variază între implementări, ceea ce impune testare atentă [83]. Există și riscuri de securitate (acces neautorizat la cheile de memorie, lipsa criptării în anumite implementări, transferuri neautorizate), de regulă atenuate prin segmentare de rețea și liste stricte de control al accesului. În ansamblu, RDMA oferă latențe foarte mici și rate de transfer ridicate pentru stocare distribuită, cu condiția unei înțelegeri aprofundate a principiilor sale de funcționare și a limitărilor de integrare [79], [80], [83].

1.3.2. RoCE și iWARP - Protocoale RDMA peste Ethernet și TCP

Această secțiune prezintă integrarea RDMA peste infrastructuri Ethernet existente și compară două tehnologii rezultate din această necesitate, RoCE și iWARP. RDMA a fost asociată inițial cu InfiniBand, însă costurile și cerințele unei infrastructuri dedicate au condus la soluții adaptate pentru Ethernet. RoCE urmărește să păstreze latența minimă specifică InfiniBand, iar iWARP valorifică stiva TCP/IP pentru a asigura fiabilitatea transferului în rețele eterogene.

RoCE are două versiuni. RoCE v1 funcționează la nivelul 2 (Ethernet), încapsulează direct datele RDMA în cadre Ethernet (EtherType 0x8915), fără IP, nu este rutabil și rămâne limitat la domeniul de broadcast. Avantajul este supraîncărcarea minimă, ceea ce permite latențe foarte mici, adecvate rețelelor locale cu debit ridicat [75]. Pentru a evita coruperea sesiunilor RDMA în lipsa mecanismelor de retransmisie, RoCE v1 necesită rețele fără pierderi (lossless) și impune control al fluxului prin Priority Flow Control (PFC) [75]. RoCE v2 operează la nivelul 3 (IP), încadrează RDMA în UDP și devine rutabil, extinzând domeniul de aplicare la rețele IP clasice. Deși antetele IP și UDP introduc o mică supraîncărcare, impactul este în general redus în rețele rapide. Totuși, dependența de comportamentul fără pierderi persistă, deoarece UDP nu asigură retransmisii, iar pierderile pot induce erori severe. În plus, apar cerințe stricte privind gestionarea congestiilor și mecanismele QoS, pentru a preveni instabilitatea în prezența traficului RDMA intens [75].

iWARP reprezintă alternativa RDMA peste TCP/IP. Protocolul păstrează modelul RDMA de acces direct la memorie, dar moștenește de la TCP fiabilitatea (retransmisii, controlul congestiilor) și capacitatea de a traversa dispozitive de rețea precum NAT și firewall-uri, fără cerințe speciale pentru infrastructură [84], [85]. Costul îl constituie o latență mai mare decât la RoCE, în special pentru transferuri mici, din cauza supraîncărcării asociate TCP (stabilirea conexiunilor, verificări de integritate, retransmisii).

Comparativ, în scenarii unde minimizarea latenței este prioritară, iar rețeaua poate fi configurată disciplinat, RoCE oferă performanțe maxime, în rețele de 100 Gbps sau mai mult atingând valori comparabile cu InfiniBand, în funcție de implementarea hardware și de parametrii de transport [74]. În schimb, iWARP se distinge prin reziliență la pierderi de pachete și simplitate operațională, putând fi implementat în infrastructuri complexe fără configurări specifice ale echipamentelor intermediare.

Operațional, RoCE cere: ajustarea PFC, configurarea QoS, politici de izolare a traficului și monitorizare continuă pentru detecția timpurie a blocajelor. iWARP este mai transparent, deoarece rulează integral peste TCP/IP. Concluzia textului este că selecția depinde de prioritățile arhitecturale, profilul aplicațiilor și infrastructura existentă: RoCE privilegiază performanța și latența minimală, iWARP privilegiază interoperabilitatea și stabilitatea, acceptând o latență superioară și o ușoară supraîncărcare datorată TCP.

1.3.3. DCB - Data Center Bridging pentru rețele fără pierderi de pachete

Ethernet a evoluat dintr-o tehnologie „best-effort” într-un mediu utilizat pentru transportul datelor critice în stocarea distribuită, însă această tranziție a necesitat mecanisme avansate de control al fluxului și de gestionare a congestiei. În acest context, Priority Flow Control (PFC) și ansamblul de standarde Data Center Bridging (DCB) devin esențiale pentru obținerea unor latențe previzibile și a unui transport fără pierderi de pachete (lossless), condiție importantă pentru fluxurile RDMA sensibile la pierderi [86], [87].

PFC, standardizat prin IEEE 802.1Qbb, depășește limitările mecanismului Ethernet PAUSE clasic. În locul opririi întregului port la congestie, PFC aplică oprirea selectivă pe clase de prioritate (conform celor opt clase definite de IEEE 802.1Q), emițând cadre de control atunci când buffer-ele pe o prioritate ating praguri prestabilite. Astfel, traficul critic (de exemplu RDMA) își menține continuitatea, în timp ce fluxurile cu prioritate mai mică sunt suspendate temporar [77]. Beneficiul este crearea unui canal lossless pentru clasele critice, însă, în distribuții de trafic inegale, PFC poate favoriza blocaje prin epuizarea buffer-elor partajate și efecte de propagare a opririlor [88].

Mecanismele de calitate a serviciului completează PFC: algoritmi de limitare și modelare a traficului, precum Weighted Round Robin, Deficit Round Robin sau Strict Priority, programează transmisia pe clase și alocă resurse proporțional cu importanța fluxurilor, menținând accesul prioritar pentru traficul critic [89].

DCB nu este un protocol unic, ci o suită de extensii care acționează concertat. Enhanced Transmission Selection (ETS, IEEE 802.1Qaz) introduce „grupuri cu lățime de bandă alocată”, permițând rezervări procentuale pe clase (de exemplu, lățime de bandă dedicată pentru RDMA), implementate direct în hardware prin scheme de programare precum DRR, WRR sau SP, ceea ce conferă predictibilitate chiar în congestie [87]. Data Center Bridging eXchange Protocol (DCBX), construit peste LLDP (IEEE 802.1AB), negociază automat între echipamente setările pentru PFC, ETS și maparea aplicațiilor sensibile la latență, reducând erorile de configurare. Există variante diferite, de la DCBX IEEE (802.1Qaz) la CEE DCBX, cu diferențe în structura și interpretarea TLV-urilor [90], [91].

Congestion Notification (CN, IEEE 802.1Qau) adaugă o componentă preventivă: la semne de congestie, switch-urile trimit CNM către surse, care își reduc ritmul de transmitere, evitând blocajele tip head-of-line produse de opriri brute. Deși standardizat, sprijinul hardware pentru CN nu este universal, ceea ce limitează aplicabilitatea în unele implementări [90]. Monitorizarea în rețele DCB necesită indicatori specifici (cadre PFC, utilizarea pe clase, frecvența CNM, niveluri de ocupare a buffer-elor), însă raportarea diferă între producători, îngreunând diagnosticarea.

Explicit Congestion Notification (ECN), la nivel IP/transport, marchează proactiv pachetele în congestie (ECN-CE), permițând receptorului să instruiască emițătorul să reducă rata. În RoCE v2, integrarea ECN contribuie la comportamentul „fără pierderi”, diminuând dependența de PFC și efectele sale colaterale [92].

În ansamblu, DCB (împreună cu PFC, ETS, DCBX, CN și ECN) a transformat Ethernet într-un substrat adecvat pentru stocare distribuită, dar reușita depinde de înțelegerea detaliată a mecanismelor, de validarea interoperabilității și de o implementare atentă a configurațiilor.

1.3.4. Integrarea RDMA în protocoalele de stocare

Secțiunea argumentează că integrarea RDMA în protocoalele de stocare nu a fost un pas incremental, ci o tranziție structurală în arhitectura subsistemelor de intrare, respectiv ieșire distribuite. Motivația provine din profilul infrastructurilor moderne, în care latențele tipice sunt de ordinul zecilor de microsecunde, volumul de operații este ridicat, iar reducerea suprasarcinii procesoarelor devine esențială. În acest cadru, protocoalele tradiționale bazate pe TCP/IP, precum NFS și SMB, nu mai satisfac cerințele actuale, deoarece implică traversări repetate ale stivei software și etape intermediare costisitoare de procesare. RDMA este, în consecință, adoptat ca fundament arhitectural, reorganizând fluxurile de date astfel încât să minimizeze latența și să reducă numărul de copii ale datelor, cu efect direct asupra eficienței.

Această schimbare impune nu doar ajustări ale logicii de transport, ci și o reevaluare a modului în care comenzile de stocare sunt mapate pe primitivele hardware ale interfețelor de rețea. Fiecare protocol trebuie să găsească un echilibru între păstrarea semanticilor existente și exploatarea avantajelor specifice RDMA, în particular accesul direct la memorie, evitarea duplicărilor de date și reducerea timpilor de răspuns. Rezultatul este apariția unei clase de protocoale hibride, compatibile cu infrastructurile deja instalate, dar optimizate pentru transport RDMA.

Din această perspectivă, textul reține ca relevante NFS over RDMA și iSCSI over RDMA, adică extensii care păstrează modelul funcțional al protocoalelor consacrate, însă îl transpun pe un canal de date cu latență redusă și procesare minimă. Sunt menționate, de asemenea, SMB Direct și NVMe over Fabrics, ca exemple semnificative ale aceleiași orientări către integrarea nativă a mecanismelor RDMA în stivă, cu scopul de a susține cerințele de performanță ale subsistemelor de stocare contemporane [66], [93], [94].

1.3.4.1. NFS over RDMA

Integrarea RDMA în NFS este standardizată prin RFC 8166, care stabilește transportul apelurilor Remote Procedure Call (RPC) peste canale RDMA fiabile, fără utilizarea stivei TCP/IP. Semantica NFS rămâne neschimbată, modificarea vizând exclusiv transportul: fluxurile TCP sunt înlocuite de operații RDMA Write și RDMA Read, transmise ca Work Requests către adaptoarele de rețea. În Linux, implementarea se bazează pe modulul xprtrdma, ce interacționează cu RPC printr-un strat abstract și împachetează apelurile în segmente XDR, iar transferul are loc între regiuni de memorie înregistrate, eliminând copiile intermediare și procesarea suplimentară în nucleu [95].

Avantajele rezultate sunt reduceri substanțiale ale latenței operațiilor cu fișiere, ca efect al evitării traversărilor TCP/IP și al accesului direct la memoria înregistrată, diminuarea semnificativă a utilizării procesoarelor, deoarece RNIC-urile preiau segmentarea și accesul la memorie, creșterea gradului de paralelism prin menținerea unui număr mare de sesiuni active cu interferență minimă, concomitent cu păstrarea compatibilității POSIX (ierarhie, permisiuni, blocări). Aceste beneficii sunt în concordanță cu specificația și au fost validate experimental în configurații adecvate [95].

Limitările vizează absența unui mecanism de retransmisie la nivel de aplicație: în medii RoCE fără control riguros al congestiei, pierderea de pachete poate invalida sesiunea RDMA. De asemenea, în implementarea standard Linux nu există un fallback automat la TCP în cazul eșecului stabilirii conexiunii RDMA, ceea ce impune politici explicite de montare sau mecanisme operaționale dedicate. Adoptarea în producție depinde de interoperabilitatea efectivă între nucleu, drivere și RNIC-uri, variabilitatea implementărilor putând produce

instabilități sub sarcină reală (de exemplu, nealinierea valorilor implicite pentru dimensiunea fragmentelor XDR sau timpii de expirare ai sesiunilor RDMA) [96].

Calitatea infrastructurii fizice este determinantă. În configurații RoCE v1/v2, menținerea avantajelor de latență presupune o rețea fără pierderi de pachete, cu mecanisme corect configurate (PFC, DCBX, ECN). În medii controlate, măsurătorile indică îmbunătățiri de 2-5 ori ale latențelor și un debit superior față de NFS/TCP în acces secvențial de citire sau scriere, concomitent cu o reducere semnificativă a utilizării CPU, rezultate obținute în condițiile unei compatibilități perfecte între RNIC-uri și drivere și ale unei rețele configurate pentru trafic determinist, fără congestie [82].

1.3.4.2. iSCSI over RDMA (iSER)

iSER (iSCSI Extensions for RDMA) standardizează integrarea RDMA în iSCSI prin RFC 7145, păstrând semantica protocolului și optimizând exclusiv transportul datelor, astfel încât mesajele de control rămân pe fluxurile iSCSI obișnuite, iar încărcătura efectivă este mutată pe operații RDMA Read și RDMA Write executate direct de adaptoarele de rețea. Separarea planului de control de planul de date scurtează traseele software și transferă segmentarea și ordonarea către hardware, eliminând trecerea prin stiva TCP/IP și copiile intermediare în memorie [97]. În Linux, implementarea folosește module dedicate la nivel de nucleu, `ib_iser` pentru inițiator și `ibisert` pentru țintă, integrate cu stiva iSCSI și subsistemul RDMA, iar descoperirea țintelor și autentificarea se desfășoară ca în iSCSI clasic. Activarea RDMA se negociază în timpul autentificării prin opțiunea RDMA Extensions, după care sesiunile intră în modul iSER și datele sunt plasate direct în zonele de memorie înregistrate pe inițiator și pe țintă [98].

Transferurile iSER se bazează pe DDP și RDMAP, care asigură verificări de integritate la nivel de segment și mențin ordonarea livrării. În consecință, HeaderDigest și DataDigest sunt dezactivate, iar în caz de eroare de transport conexiunea se închide, recuperarea făcându-se prin mecanismele de reconectare iSCSI, nu prin retransmisii de PDU. Pe partea de implementare, `ib_iser` descarcă comenzile către adaptorul de rețea, `ibisert` expune suportul în subsistemul LIO, iar administrarea se face cu `iscsiadm` și `targetcli`, ceea ce ușurează adoptarea în centrele de date fără schimbări operaționale majore.

Beneficiile principale derivă din zero-copy și din offload-ul masiv în hardware, cu latențe reduse ale tranzacțiilor, utilizare mai mică a CPU și capacitate sporită de paralelizare a comenzilor. Protocolul este independent de transportul RDMA utilizat, fiind compatibil cu InfiniBand, RoCE și iWARP [99]. În practică, RoCE solicită rețele fără pierderi, configurate pentru comportament determinist, altfel sesiunile pot deveni instabile deoarece RDMA nu definește retransmisii la nivel de aplicație. Pentru scenariile IP, securizarea se aliniază recomandărilor din RFC 7146, cu posibilitatea folosirii IPsec pentru criptare și autentificare, iar standardele RDDL și RDMAP prevăd protecția accesului la buffer-ele înregistrate.

Stabilitatea depinde de interoperabilitatea dintre RNIC-uri, drivere, firmware și versiunea de nucleu, inclusiv de alinierea parametrilor critici precum MTU și temporizările. iSER nu oferă revenire automată la TCP, așadar politicile de comutare trebuie definite explicit. În medii HPC și în centre de date cu multe sesiuni iSCSI, descărcarea operațiilor de date către hardware reduce întârzierile și eliberează procesoarele pentru aplicații, iar într-o infrastructură corect configurată iSER oferă latențe mai mici, consum mai redus de CPU și o scalabilitate superioară, menținând în același timp modelul de comandă SCSI și instrumentele de administrare consacrate.

1.4. Considerații de securitate

1.4.1. Elemente de securitate la nivelul stocării

Sistemele de stocare au devenit componente critice în infrastructurile actuale, pe fondul creșterii continue a serviciilor digitale, a volumelor de date și a cerințelor de acces rapid. Arhitecturile au evoluat către soluții distribuite și mecanisme de scalare, iar securitatea este tratată ca fundament, la nivel hardware și software, cu cerințe de confidențialitate, integritate, disponibilitate, trasabilitate și reziliență, dar și cu obligații legale și economice impuse de

reglementări și de riscurile incidentelor de securitate [29]. Modelul CIA (Confidentiality, Integrity, Availability) oferă cadrul conceptual: confidențialitatea protejează datele împotriva accesului neautorizat, integritatea garantează detectarea oricărei alterări neautorizate, disponibilitatea asigură accesul la date ori de câte ori sunt necesare, inclusiv în condiții de avarii sau atacuri [100]. Măsurile concrete depind de tipul infrastructurii de stocare, pe blocuri, fișiere sau obiecte, iar capitolul detaliază componentele și practicile folosite pentru reducerea riscurilor de securitate cibernetică.

1.4.1.1. Confidențialitatea datelor

Protecția confidențialității în arhitecturile de stocare se bazează pe criptare, cu accent pe criptarea datelor la repaus, unde criptarea integrală a discurilor (Full Disk Encryption, realizată pe dispozitive Self-Encrypting Drives) mută operațiile criptografice în hardware și reduce sarcina asupra procesorului. Complementar, criptarea la nivel de fișier sau obiect oferă control granular și costuri de performanță mai mici, fiind frecvent integrată în API-urile platformelor cloud. Gestionarea cheilor este asigurată de sisteme KMS, care generează chei cu entropie ridicată, le stochează în module HSM, aplică rotație periodică și mențin auditări ale accesului, pentru respectarea politicilor și a reglementărilor [101], [102]. Implementarea defectuoasă a managementului cheilor anulează beneficiile criptării, motiv pentru care, în industriile reglementate, sunt utilizate HSM certificate, de exemplu conform FIPS 140-2 Level 3 [103].

1.4.1.2. Integritatea datelor

Integritatea datelor urmărește prevenirea modificărilor neautorizate sau accidentale și detectarea rapidă a acestora, prin mecanisme precum sumele de control, care sunt recalcule la citire și comparate cu valorile salvate, structurile Merkle Tree, care permit verificări eficiente pe seturi mari de date, și tehnologia WORM, care blochează suprascrierea după prima înregistrare [104]. Un risc major este „alterarea silențioasă” a datelor, când biți se modifică fără erori vizibile pentru sistem sau aplicații [105]. Sisteme de fișiere avansate, de tipul ZFS și Btrfs, folosesc sume de control pe bloc și pot reface conținutul din copii redundante atunci când acestea există, în absența acestor mecanisme corupțiile pot rămâne nedetectate și pot compromite întregul set de date [104].

1.4.1.3. Disponibilitatea datelor

Disponibilitatea în sistemele de stocare înseamnă acces continuu la date, chiar în prezența defecțiunilor hardware, erorilor software sau a altor incidente. Reducerea riscului de indisponibilitate se realizează, în primul rând, prin redundanță: la nivel hardware (RAID, pentru refacerea datelor la defectarea unui disc) și la nivel logic (replicare pe discuri ori noduri diferite, inclusiv geografic, pentru continuitatea accesului la informații) [106]. Copiile instantanee (snapshots) permit revenirea rapidă la o stare anterioară și sunt utile în recuperarea după ștergeri accidentale sau atacuri de tip ransomware. Tehnicile de preluare automată a funcționalităților (failover) asigură transferul rapid al serviciilor către un sistem sănătos la căderea unei componente. Planurile de recuperare după dezastru (Disaster Recovery) stabilesc frecvența copiilor de siguranță, locațiile de stocare și ordinea repunerii în funcțiune a serviciilor, fiind esențială testarea periodică a acestor proceduri. În arhitecturi critice, și rețeaua trebuie să fie redundantă (legături multiple și rerutare transparentă), deoarece o conectivitate slabă poate provoca întârzieri sau întreruperi ale accesului la date [106]. În final, disponibilitatea rămâne o prioritate operațională, fiind direct percepută de utilizatori: dacă datele nu pot fi accesate la nevoie, celelalte măsuri de securitate își pierd relevanța.

1.4.1.4. Controlul accesului

Controlul accesului definește cine poate citi, modifica sau șterge date și este indispensabil pentru eficiența celorlalte măsuri de securitate. Modelul clasic bazat pe permisiuni POSIX este adecvat în scenarii simple, dar devine dificil de administrat la scară în infrastructuri complexe [107]. ACL-urile aduc granularitate la nivel de utilizator și grup, însă cresc complexitatea operațională. Modelele de tip MAC, precum SELinux și AppArmor, impun politici la nivel de sistem și limitează impactul compromiterii unor servicii. În medii multi-

utilizator și multi-aplicație, controlul accesului se integrează cu sisteme centralizate de identitate, de exemplu LDAP sau Active Directory, pentru administrare unificată a drepturilor. Autentificarea și autorizarea se bazează pe mecanisme consacrate, precum Kerberos sau OAuth, pentru aliniere cu restul serviciilor din infrastructură. Jurnalizarea accesului rămâne esențială, deoarece furnizează trasabilitate și suport pentru investigații și prevenție în cazul incidentelor de securitate [107].

1.4.1.5. Protecția împotriva ransomware

Atacurile ransomware, care criptează fișierele locale sau din volume partajate și solicită răscumpărare, pot provoca întreruperi și pierderi financiare dacă nu există măsuri de protecție adecvate [108]. Limitarea impactului se bazează pe copii sau imagini instantanee (snapshots) pentru revenire rapidă, însă acestea nu înlocuiesc copiile de siguranță externe și pot fi vulnerabile dacă sunt șterse ori modificate de malware. Protecția se completează cu setări de imutabilitate ale fișierelor sau volumelor (de exemplu, attribute speciale în Linux) care blochează modificările neautorizate. Sistemele moderne includ monitorizare a comportamentului aplicațiilor și utilizatorilor pentru a detecta rapid acțiuni suspecte, uneori bazată pe algoritmi de învățare automată [109], [110]. Este necesară o strategie diversificată de backup, cu copii păstrate și pe medii externe separate fizic sau logic (air-gapping), precum și criptarea copiilor de siguranță. Jurnalizarea detaliată sprijină răspunsul la incidente, permițând identificarea sursei, estimarea extinderii și a zonelor compromise, precum și înțelegerea modului de pătrundere și propagare a malware-ului [107]. Protecția eficientă împotriva ransomware necesită combinații de măsuri tehnice, politici de securitate și procese operaționale bine definite și verificate.

1.4.1.6. Securitatea în medii virtualizate și cloud

Migrarea aplicațiilor către platforme virtualizate și servicii cloud crește complexitatea controlului accesului și a protecției datelor în medii în care mai mulți clienți folosesc aceeași infrastructură. Izolarea spațiilor de nume rămâne principiul de bază: fiecare organizație primește volume ori obiecte de stocare proprii, separate prin politici la nivelul hipervizorului și prin mecanismele sistemelor de stocare, inclusiv în stocarea pe obiecte, unde separarea se realizează prin structuri și chei distincte de acces [111].

Criptarea individuală pe client, aplicată direct fișierelor sau fragmentelor de date, utilizează chei unice gestionate de servicii specializate, care acoperă întreg ciclul de viață al cheilor, de la generare și păstrare sigură până la înlocuire sau revocare [102], [111]. În paralel, hipervizoarele reduc riscul atacurilor de tip escape prin politici stricte de izolare și monitorizare a accesului la resursele de stocare, completate de profiluri de securitate care limitează operațiile permise fiecărei instanțe.

În orchestratoare precum Kubernetes, accesul la volume persistente este controlat prin Container Storage Interface, care permite politici explicite pentru modul de montare, capacitate și comportamentul volumelor la ștergerea aplicației [112]. La nivel de API, accesul la stocare se protejează prin validarea riguroasă a autenticității cererilor, autorizare, semnare și criptare a solicitărilor, precum și limitarea ratei de acces, pentru a reduce riscul de reluare a tranzacțiilor sau blocare a serviciilor prin inundare [113].

Rotirea periodică a cheilor și revocarea rapidă a certificatelor sau a token-urilor limitează fereastra de expunere în cazul compromiterii unui cont. În plus, politicile de securitate la nivel de rețea, inclusiv segmentarea traficului și grupurile de securitate, restricționează fluxurile permise între resurse, astfel încât un container compromis să nu poată interoga alte volume de stocare din infrastructură [113].

Împreună, aceste măsuri tehnice și organizatorice furnizează un cadru de protecție a datelor în medii cu resurse de stocare partajate, inclusiv atunci când accesul are loc peste rețele accesibile publicului.

1.4.2. Considerații de securitate în rețele RDMA

Adoptarea RDMA reduce latențele și consumul de resurse din procesoare în stocarea distribuită, însă introduce riscuri specifice, deoarece expune controlat regiuni de memorie ale

aplicațiilor. Cheile de acces asociate memoriei înregistrate sunt transmise pe canalul de control, iar compromiterea lor permite citirea sau scrierea directă în memorie, fără protecțiile uzuale din sistemul de operare. InfiniBand nu oferă criptare în tranzit, RoCE moștenește expunerea nivelului de legătură și este vulnerabil la spoofing, injection și replay, iWARP poate folosi TLS sau IPsec, cu costul unei supraîncărcări suplimentare și al unei creșteri minime a latenței.

Filtrarea și inspecția standard nu se aplică fluxurilor RDMA, care ocolesc stiva software de rețea, de aceea protecția trebuie mutată în hardware și în rețeaua fizică, prin izolări la nivel 2 și politici stricte, însă suportul echipamentelor este înegal. În RoCEv2, folosirea UDP/IP facilitează interceptarea și reinjectarea pachetelor în lipsa criptării sau a semnării digitale, iar utilizarea porturilor fixe crește suprafața de atac. În medii multi-tenant, izolarea fină este dificilă, motiv pentru care se recurge la mecanisme hardware și arhitecturale dedicate, de tip P_Key în InfiniBand sau politici DCB și marcaje VLAN în RoCE, care necesită proiectare și operare riguroasă. În virtualizare, partajarea adaptoarelor prin SR-IOV poate genera scurgeri de date sau preluarea QP-urilor dacă separarea contextelor nu este completă.

Adoptarea RDMA trebuie corelată cu politicile de securitate și cu cerințele de izolare, ținând cont de limitele resurselor hardware pentru memoria înregistrată, de vizibilitatea redusă și de compromisurile între securitate și performanță, precum și de gradul de suport al ecosistemului software. Implementat și configurat corect, RDMA aduce câștiguri consistente, dar fără o abordare riguroasă acestea pot fi anulate de limitările arhitecturale și de complexitatea operațională [114].

1.5. Concluzii intermediare

Capitolul definește modelele de servicii (IaaS, PaaS, SaaS) și tipurile de implementare (public, privat, hibrid), cu aplicații directe în mediul universitar: laboratoare virtuale, VDI pe cloud privat și scenarii didactice automatizate pentru securitate, rețele, programare paralelă și DevOps. Sunt evidențiate beneficiile privind accesul, flexibilitatea și personalizarea pe platforme open-source, precum și necesitatea competențelor operaționale și a evaluării continue a performanței în funcție de hardware, rețea și stocare.

Componenta de stocare este structurată pe cele două paradigme principale (bloc și fișier), cu impact asupra IOPS, latenței, rezilienței și complexității operaționale. Pentru stocarea pe bloc sunt analizate dimensiunea blocului și adâncimea cozilor, inclusiv capabilitățile NVMe și efectele asupra paralelismului și latenței, iar alegerea parametrilor se face prin teste reprezentative pentru sarcina de lucru. La nivel de fișiere sunt explicate rolurile și structurile interne, apoi câteva implementări: ext4 (extents, moduri de jurnalizare, attribute extinse și ACL), XFS (orientat spre fișiere mari), ZFS (validarea integrității, instantanee, compresie și auto-remediere, cu costuri de resurse, ARC, și particularități legate de eliberarea spațiului). Pentru acces concurent pe volum partajat este detaliat OCFS2 (jurnalizare per nod, consistență POSIX, distribuția metadatelor fără punct unic de defecțiune). În zona sistemelor de fișiere de rețea este descris NFS, fiind menționate blocările, problemele de consistență și configurațiile care influențează performanța. Sunt tratate tehnologiile avansate de transport și rețea (RDMA, RoCE, iWARP, DCB), cu accent pe reducerea latenței și descărcarea procesoarelor, precum și integrarea RDMA în protocoalele de stocare (NFS over RDMA și iSCSI over RDMA).

Secțiunea de securitate abordează modelul CIA, controlul accesului (permisiuni POSIX, ACL, MAC, SELinux, AppArmor), integrarea cu sisteme de identitate centralizată, autentificarea și autorizarea, politici pentru NFS și segmentarea la nivel de rețea (grupuri de securitate), precum și riscuri specifice RDMA, inclusiv gestionarea cheilor de acces la regiunile de memorie. Capitolul se încheie cu trimiterea către analiza platformelor de management al cloud-urilor și o comparație, pe mai multe criterii, a celor mai reprezentative trei platforme utilizate în prezent.

Capitolul II - Platforme cloud open-source pentru infrastructuri virtualizate

Capitolul introduce platformele open-source de tip Cloud Management Platform (CMP) drept instrumente principale pentru IaaS, care oferă medii dedicate pentru rularea și administrarea mașinilor virtuale (VM), cu flexibilitate, scalabilitate și eficiență a costurilor. Aceste soluții evită dependența de un singur furnizor sau de hardware specializat, oferă control deplin asupra datelor și permit personalizarea infrastructurii. Sunt menționate OpenStack, Apache CloudStack și OpenNebula, fiecare cu arhitectură și filozofie proprii, dar cu același obiectiv, rularea VM-urilor într-un context IaaS și managementul infrastructurii.

2.1. OpenStack

OpenStack este o platformă software open-source pentru construirea și gestionarea infrastructurilor cloud de tip IaaS, inițiată în 2010 ca parteneriat NASA și Rackspace Hosting, cu obiectivul de a furniza o platformă flexibilă, scalabilă și complet open-source pentru administrarea centrelor de date. În prezent este dezvoltată sub umbrela OpenInfra Foundation, cu contribuții din partea a sute de organizații și mii de dezvoltatori, fiind implementată în centre de date enterprise, instituții academice și la mari furnizori de cloud public.

Arhitectura este modulară, compusă din servicii interconectate cu responsabilități precise, care comunică prin API-uri REST și folosesc brokeri de mesaje, de exemplu RabbitMQ sau Qpid, pentru coordonarea operațiilor asincrone, ceea ce permite scalarea pe orizontală. Serviciile principale includ: Nova (compute), Neutron (rețea), Glance (imagini), Cinder (stocare la nivel de bloc), Swift (stocare la nivel de obiecte), Keystone (autentificare și autorizare), Horizon (interfață web), Heat (orchestrare), Ceilometer și Gnocchi (telemetrie), Placement (alocare și urmărire a resurselor).

OpenStack oferă suport extins pentru hipervizoare, incluzând KVM împreună cu QEMU, VMware vSphere și vCenter, Hyper-V, Xen și XenServer, LXC și LXI, prin drivere specifice, denumite virt drivers. În contextul KVM sunt utilizate imagini QCOW2 și RAW, formatul QCOW2 fiind preferat pentru instantanee și compresie. Integrarea cu VMware se face prin VMwareVCDriver și VMwareAPIVMwareVCDriver, cu imagini VMDK. Migrarea între hipervizoare diferite nu este posibilă direct, necesitând conversii de format și, de regulă, ajustări de configurare, OpenStack neoferind interoperabilitate automată.

Modelul de stocare cuprinde Cinder pentru volume persistente la nivel de bloc cu backend-uri precum LVM, Ceph RBD, NFS, iSCSI, Fibre Channel și soluții proprietare, cu gestionarea ciclului de viață al volumelor, inclusiv instantanee și migrare acolo unde este posibil, Swift pentru stocare distribuită de obiecte cu replicare și toleranță la defecte, fără a impune RAID la scară mare, Glance pentru gestionarea imaginilor discurilor virtuale stocate local sau în backend-uri externe precum Swift, Ceph ori Amazon S3 prin drivere, în formate QCOW2, RAW, VMDK și altele compatibile cu hipervizorul țintă.

Neutron permite rețele virtuale izolate pe client, topologii complexe și servicii de rețea precum VLAN 802.1Q, rețele suprapuse VXLAN sau GRE, integrare cu SDN, inclusiv OpenDaylight, OVN, Nuage Networks și Open vSwitch, rutere virtuale, NAT, DHCP, firewall, VPN, Quality of Service, asigurând izolare multiutilizator și politici de securitate granulare. Planificatorul Nova decide plasarea pe baza capacităților CPU și RAM, a politicilor de plasare, afinitate sau anti-afinitate și a filtrelor personalizate, iar Placement gestionează alocarea detaliată a resurselor fizice.

Keystone oferă autentificare bazată pe token-uri, integrare cu LDAP și Identity Federation, de exemplu SAML și OpenID Connect, controlul accesului bazat pe roluri (RBAC). Horizon furnizează interfața web oficială, iar interacțiunea este posibilă și prin CLI, SDK-uri și API REST. Heat orchestrează stive descrise în șabloane YAML (Heat Orchestration Templates), susține scalarea și actualizările treptate (rolling updates). Telemetria a evoluat prin Ceilometer, Gnocchi și Panko, integrate frecvent cu Prometheus, Grafana sau Zabbix. Platforma este extensibilă prin plugin-uri ML2, drivere de stocare, resurse Heat și integrare SDN și NFV, este distribuită sub licența Apache 2.0 și este adoptată pe scară largă în telecom,

HPC și cloud public, poziționându-se ca o soluție matură pentru infrastructuri cloud de mari dimensiuni și sarcini de lucru critice.

2.2. CloudStack

Proiectul CloudStack [116] a fost inițiat de Cloud.com, preluat de Citrix în 2011 și donat în 2012 către Apache Software Foundation, trecând prin Apache Incubator și fiind ulterior promovat ca proiect de nivel superior, cu guvernare și licențiere Apache. Platforma este multiutilizator, cu izolare a resurselor între organizații, folosind o ierarhie logică de regiuni, zone, pods, clustere și hosts, care mapează fidel topologia fizică și virtuală. Arhitectura este structurată pe niveluri și cuprinde Management Server, componenta de orchestrare și control, Database Server pentru starea infrastructurii și metadate, Hypervisor Hosts pentru execuția mașinilor virtuale, Secondary Storage pentru șabloane, imagini ISO și copii instantanee. Management Server concentrează logica de control, regulile de business, autentificarea și interfațarea cu utilizatorii și API-urile.

CloudStack suportă mai multe hipervizoare, inclusiv KVM, VMware vSphere și vCenter, XenServer și XCP-ng, Hyper-V, cu mențiunea că suportul Hyper-V nu mai este întreținut în versiunile recente. Formatele de imagini utilizate includ QCOW2 și RAW pentru KVM, VMDK pentru VMware, VHD pentru XenServer și XCP-ng. Migrarea transparentă între hipervizoare diferite nu este disponibilă, schimbarea hipervizorului necesitând conversii de format și ajustări de configurare, însă definirea de șabloane distincte pe hipervizor permite infrastructuri mixte, fără transfer direct al instanțelor fără intervenții suplimentare.

Modelul de stocare este împărțit în Primary Storage, care găzduiește discurile virtuale active și susține backend-uri locale sau partajate, inclusiv NFS, iSCSI, Fibre Channel, Ceph RBD, stocarea partajată fiind preferată deoarece permite migrarea în timp real și redundanță, și Secondary Storage, depozit centralizat pentru șabloane, imagini ISO și copii instantanee, uzual pe NFS, cu integrare posibilă cu Amazon S3, Swift sau Ceph RGW. Platforma oferă Storage Pools, locale sau partajate, și Storage Tags, prin care administratorii direcționează volumele către grupuri de discuri adecvate, de exemplu SSD sau HDD.

Serviciile de rețea pot fi administrate sub forma a două modele, conectivitate de bază, cu același segment la nivelul 2, și conectivitate avansată, cu rețele izolate pe organizații folosind VLAN, VXLAN sau overlay. În modul avansat sunt integrate rutere virtuale pentru NAT, DHCP, firewall și distribuirea încărcării, echipamente externe precum Juniper SRX, Palo Alto Networks sau Cisco ASA, controlere SDN precum OpenDaylight sau Nuage VSP, cu posibilitatea de marcare VLAN 802.1q. Planificatorul plasează instanțele în funcție de utilizarea resurselor, politici de afinitate și anti-afinitate, constrângeri topologice și disponibilitate, permițând politici personalizate.

Modelul de securitate include Domenii, Conturi și Proiecte, controlul accesului bazat pe roluri și integrare LDAP. Integrarea cu clouduri publice este posibilă prin mecanisme precum CloudBridge și stocarea în S3, însă extinderea resurselor în clouduri publice nu este nativ avansată. Interacțiunea cu platforma se face prin interfața web, linia de comandă CloudMonkey și API. Extensibilitatea este asigurată prin plugin-uri integrate în serverul de management și webhooks, acestea din urmă fiind mecanisme de notificare, iar extinderea comportamentului dincolo de notificări se realizează prin plugin-uri. CloudStack este distribuit sub licența Apache 2.0, susținut de o comunitate activă, utilizat în implementări de anvergură, inclusiv în mediul academic, unde este menționat ca potrivit pentru instituții de învățământ datorită cerințelor hardware moderate și interfeței de administrare intuitive [16].

2.3. OpenNebula

OpenNebula este o platformă software open-source pentru managementul infrastructurilor virtualizate de tip IaaS, utilizată în cloud-uri private, hibride și publice în institute de cercetare, instituții guvernamentale și companii private. Platforma se remarcă prin simplitatea implementării, flexibilitatea arhitecturii și integrarea cu KVM, VMware, LXC sau furnizori de infrastructură publică, fiind adecvată pentru medii enterprise, proiecte de cercetare și servicii comerciale.

Arhitectura este minimalistă și modulară, cu obiectivul reducerii complexității operaționale, fără a compromite flexibilitatea și scalabilitatea. Componenta centrală este oned, care menține starea globală a infrastructurii și orchestrează operațiile asupra mașinilor virtuale, imaginilor de disc, rețelelor virtuale și utilizatorilor, comunicând cu subsistemele prin interfețe API clar definite. Separarea între controlul logic și infrastructura fizică permite integrarea într-o singură instanță operațională a unor tehnologii hardware și software diverse, principiul de bază fiind interoperabilitatea pentru administrarea infrastructurilor hibride și eterogene.

OpenNebula suportă KVM și VMware vCenter și, în anumite scenarii, containere LXD. În KVM sunt gestionate imagini QCOW2 și RAW, formatul QCOW2 fiind preferat pentru copii instantanee, compresie și alocare dinamică, iar formatul RAW pentru performanțe predictibile. Integrarea cu VMware utilizează formatul VMDK, exclusiv prin vCenter. Migrarea între hipervizoare diferite necesită conversia formatelor și ajustări la nivelul configurației hardware virtuale, procesul nefiind complet automatizat. Interoperabilitatea transparentă între hipervizoare nu este disponibilă.

Subsistemul de stocare este flexibil și include, pentru KVM, tipuri de Datastore pe sistem de fișiere, LVM, Ceph cu replicare și toleranță la defecte, respectiv Datastore partajate accesibile simultan prin rețea. În medii VMware, platforma interacționează cu datastore-urile definite în vCenter, de tip VMFS sau NFS, operațiile fiind executate prin API-urile vCenter. Imaginile de disc sunt resurse gestionate unitar, cu operații de clonare, instantanee acolo unde backend-ul permite, gestiune de versiuni și ștergere programată a resurselor neutilizate.

Componenta de rețelistică susține rețele virtuale cu etichetare VLAN, integrare cu Open vSwitch, rețele suprapuse prin VXLAN și Geneve și scenarii SDN. Sunt disponibile rutere virtuale care furnizează NAT, DHCP și distribuirea încărcării, permițând comunicarea controlată între rețele izolate și rețele externe. Planificatorul alocă instanțele pe noduri fizice în funcție de capacitatea CPU, memoria RAM disponibilă, starea și performanța backend-urilor de stocare și politici de priorizare, afinitate și anti-afinitate.

Modelul de securitate este bazat pe RBAC, cu roluri și grupuri de utilizatori, integrare cu LDAP și Microsoft Active Directory, precum și suport pentru autentificare multi-factor. Versiunile recente includ Edge Clusters pentru scenarii de edge computing și funcționalități de cloud bursting către Amazon EC2 și Microsoft Azure, administrate prin același plan de control. Interfața web Sunstone oferă administrarea completă a infrastructurii, iar Service Templates și Application Flows susțin aplicații multi-tier cu dependențe, scalare și acțiuni de remediere. Suportul pentru hooks permite automatizarea prin scripturi la evenimente relevante. OpenNebula este distribuită sub licența Apache 2.0 și are, pe lângă versiunea open-source, ediții comerciale cu funcționalități și servicii suplimentare.

2.4. Diferențe și similitudini între OpenStack, CloudStack și OpenNebula

Secțiunea compară OpenStack, CloudStack și OpenNebula, evidențiind obiectivele comune și diferențele tehnice care le afectează performanța, flexibilitatea și integrarea în medii IT complexe. Maturitatea proiectelor, arhitectura modulară și sprijinul comunităților sau al producătorilor influențează capacitatea de a răspunde cerințelor comerciale, educaționale și de cercetare. Diferențele apar la nivelul mecanismelor de orchestrare, al administrării rețelei și stocării, precum și în compatibilitatea cu hipervizoare și formate de imagini ori în instrumentele de integrare. OpenStack este caracterizat de un ecosistem vast și opțiuni de personalizare, CloudStack de simplitate și implementare rapidă, OpenNebula de ușurință în utilizare și consum redus de resurse.

Comparațiile din literatura de specialitate vizează în principal OpenStack, CloudStack, OpenNebula, precum și uneori Eucalyptus și Nimbus [24], [119], [120]. OpenStack este frecvent prezentat ca soluție extensibilă, cu arhitectură modulară și susținere largă a industriei și a comunității [21], [121], însă instalarea și operarea pot fi complexe, necesitând eforturi de configurare și administrare considerabile [122], [123]. CloudStack este descris ca platformă stabilă, cu arhitectură mai simplă și orientare pragmatică, fără a compromite funcționalitatea [12], [124], [125], evaluările comparative indicând performanțe competitive, în special la gestionarea rețelei și a resurselor virtualizate [122], [126]. OpenNebula este menționată pentru simplitate și ușurința configurării, adecvată infrastructurilor academice și private de dimensiuni

moderate [21], [127], în timp ce Eucalyptus este analizată prin compatibilitatea cu API-urile Amazon EC2 și S3, utilă pentru infrastructuri hibride [120].

Din perspectiva performanței, lucrările evidențiază variații semnificative ale timpilor de instanțiere a mașinilor virtuale, ale utilizării CPU și memoriei, precum și ale capacității de scalare [21], [22], accentuând importanța mecanismelor scalabile de rezervare automată a resurselor în centre de date mari [128]. Securitatea, interoperabilitatea și extensibilitatea apar ca factori de diferențiere în integrarea platformelor în arhitecturi cloud private sau hibride [24], [121], avantajele de integrare extensivă presupunând adesea costuri operaționale crescute.

Concluzia subliniază că alegerea unei platforme open-source IaaS trebuie să echilibreze complexitatea cu capabilitățile tehnice și cu contextul organizațional, studiile păstrându-și relevanța în pofida absenței unor referințe explicite la versiuni recente. Pentru analiza detaliată sunt avute în vedere versiunile curente la momentul redactării, respectiv OpenStack 2025.1 Epoxy [115], CloudStack 4.20.1 LTS [117], OpenNebula 7.0 Phoenix [129].

2.4.1. Arhitectura și modelul de operare

Sunt comparate arhitecturile OpenStack, CloudStack și OpenNebula, accentuând modul în care alegerile de proiectare influențează scalabilitatea, complexitatea operațională și integrarea. OpenStack [115] se bazează pe o arhitectură profund modulară și distribuită, în care servicii precum Nova, Neutron, Cinder, Keystone și Glance operează independent, expuse prin interfețe REST proprii. Coordonarea interservicii este realizată prin brokeri de mesaje, de regulă RabbitMQ sau Qpid, ceea ce permite orchestrarea asincronă și decuplarea componentelor în scenarii de scalare ridicată. Persistența datelor și a metadatelor infrastructurii este asigurată prin baze de date SQL, uzual MySQL, MariaDB sau PostgreSQL. Din punct de vedere operațional, serviciile sunt proiectate să fie stateless, logica fiind separată de datele persistente, fapt care facilitează replicarea pe mai multe noduri. Această modularitate introduce însă un nivel semnificativ de complexitate, mai ales la actualizări și integrarea de funcționalități noi, motiv pentru care OpenStack este asociat frecvent cu infrastructuri enterprise de mari dimensiuni.

CloudStack [117] adoptă o arhitectură pe mai multe straturi, orientată spre reducerea dificultății de implementare și operare. Nucleul logic este Management Server, care orchestrează întregul mediu și interacționează cu un Database Server, de regulă MySQL sau MariaDB, unde sunt păstrate datele persistente despre infrastructură, utilizatori și configurații. Execuția mașinilor virtuale are loc pe nodurile de găzduire, iar Secondary Storage este utilizat pentru șabloane, imagini ISO și copii instantanee. Platforma introduce o ierarhie logică de regiuni, zone, pod-uri, clustere și gazde, ce reflectă topologia centrului de date și permite administrare granulară. Spre deosebire de OpenStack, CloudStack nu folosește brokeri de mesaje pentru comunicarea internă, bazându-se pe interacțiuni directe prin API și în baza de date, ceea ce simplifică instalarea și operarea, dar poate limita scalarea la dimensiuni foarte mari, unde replicarea și balansarea sunt mai restrictive.

OpenNebula adoptă o arhitectură centralizată și minimalistă, concepută pentru management eficient cu supraîncărcare operațională redusă. Orchestrarea este realizată de serviciul principal de pe serverul de management, care gestionează mașinile virtuale, imaginile de disc și rețelele virtuale. Comunicarea cu hipervizoarele se face direct, prin conexiuni SSH sau API-urile acestora, fără brokeri de mesaje intermediari. Persistența datelor utilizează SQLite în scenarii de mică amploare sau MySQL și MariaDB pentru infrastructuri mai complexe, iar interfața web integrată permite administrarea completă fără componente externe suplimentare. Extensibilitatea este posibilă prin drivere externe și mecanisme precum OneFlow. Scalabilitatea orizontală este mai limitată decât în cazul OpenStack, însă simplitatea și consumul redus de resurse pot compensa aceste limitări în numeroase contexte.

Se poate observa astfel că OpenStack este proiectat pentru centre de date extinse, fiind capabil să susțină scenarii complexe și scalabilitate ridicată, dar presupune o complexitate operațională considerabilă. CloudStack reprezintă un compromis între flexibilitate și simplitate, fiind mai ușor de implementat și gestionat decât OpenStack, dar cu limitări în medii de foarte mare amploare. OpenNebula, prin arhitectura sa centralizată și minimalistă, se adresează în special organizațiilor care preferă soluții rapide, eficiente și cu necesar de efort

operațional redus, sacrificând însă posibilitățile extinse de scalabilitate orizontală oferite de platforme precum OpenStack.

2.4.2. Suportul pentru hipervizoare

OpenStack oferă cel mai extins spectru de hipervizoare, fiind proiectat încă de la început ca hipervizor agnostic. În versiunea Epoxy 2025.1, integrarea rămâne solidă pentru KVM cu QEMU, VMware vSphere cu vCenter, Hyper-V, XenServer cu XCP-ng și containere LXC, LXD, prin drivere dedicate Nova. KVM este hipervizorul de referință în comunitatea OpenStack, recomandat în majoritatea scenariilor de producție și pentru cloud public. Integrarea cu VMware vSphere permite utilizarea clusterelor administrate de vCenter, oferind interoperabilitate pentru organizații care dețin deja acest ecosistem. Suportul pentru XenServer, XCP-ng este oficial, dar mai puțin popular în implementări de anvergură. În cazul Hyper-V, suportul există, însă este limitat și eliminat în numeroase distribuții comerciale. LXC, LXD pot fi utilizate, deși în producție sunt mai rar întâlnite. Suportul pentru acceleratoare hardware, GPU și FPGA, este consolidat prin Cyborg, cu acoperire multiplă de hipervizoare, în special pentru KVM și VMware.

CloudStack 4.20.1 LTS susține o gamă variată de hipervizoare, dar mai restrânsă decât OpenStack. Integrarea stabilă include KVM, VMware prin vSphere, vCenter și XenServer, XCP-ng, acestea fiind opțiunile principale în producție. În context VMware, platforma gestionează mașinile virtuale prin vCenter și poate utiliza șabloane și profiluri hardware definite în VMware. Pentru XenServer, XCP-ng există suport aproape complet, folosit în numeroase implementări pentru stabilitate și costuri reduse. Suportul pentru Hyper-V a existat în versiunile vechi, dar nu mai este recomandat și nu mai beneficiază de întreținere activă. Spre deosebire de OpenStack, CloudStack nu include suport pentru LXC, LXD și nu tratează containerele ca alternativă internă la mașini virtuale. Pentru acceleratoare, platforma poate gestiona conectarea directă a GPU, passthrough și vGPU în XenServer și VMware, însă profilurile vGPU sunt administrate la nivelul hipervizorului, nu orchestrat intern.

OpenNebula 7.0 Phoenix adoptă o abordare concentrată pe hipervizoarele cele mai răspândite. Suportul oficial acoperă KVM, VMware vCenter și LXD. KVM este hipervizorul principal, cu integrare profundă pentru stocare, rețea și acceleratoare, GPU, vGPU. Integrarea cu VMware este completă, permițând gestionarea infrastructurilor existente din vCenter, cu păstrarea șabloanelor și politicilor, însă multe funcții, de exemplu managementul vGPU, rămân administrate prin vCenter. Suportul pentru LXD aduce integrarea simultană a mașinilor virtuale și containerelor sub aceeași interfață. Pentru acceleratoare, există suport oficial de vGPU în KVM, introdus în versiunile recente, cu definirea profilurilor la nivel de șablon de mașină virtuală.

OpenStack rămâne singura platformă care orchestrează direct profilurile vGPU prin Cyborg, oferind control complet asupra resurselor din acceleratoare. CloudStack are facilități mai restrânse, fără suport pentru containere și fără management intern al profilurilor vGPU, dar cu comportament stabil în medii VMware și Xen. OpenNebula a adăugat suport nativ de vGPU pentru KVM și containere LXD, însă nu permite utilizarea hipervizoarelor Hyper-V și Xen. Din perspectiva compatibilității cu hipervizoare, OpenStack are cea mai largă acoperire, CloudStack ocupă un loc intermediar, iar OpenNebula se concentrează pe un set mai restrâns, compensând prin simplitate operațională.

2.4.3. Managementul sistemelor de stocare

OpenStack (Epoxy 2025.1) prezintă cel mai amplu ecosistem de integrare a stocării la nivel de blocuri, obiecte și sisteme de fișiere. Componenta centrală pentru blocuri este Cinder, care administrează volume persistente atașabile/detașabile pentru instanțele Nova și oferă suport pentru multiple backend-uri (Ceph RBD, LVM, iSCSI, Fibre Channel, NFS și soluții comerciale precum NetApp, Dell EMC, HPE 3PAR, Huawei). Funcționalități precum alocare subțire, copii instantanee și clonare sunt disponibile nativ în multe backend-uri. Pentru imaginile de disc, Glance poate utiliza stocări locale, NFS, respectiv sisteme de obiecte (Swift), Amazon S3 sau Ceph RGW. Swift furnizează stocare de obiecte scalabilă, cu containere pentru fișiere, imagini și log-uri. Integrarea cu Ceph acoperă Cinder, Glance și Nova (discuri

temporare), ceea ce permite folosirea Ceph drept soluție unificată, cu performanță și reziliență ridicate, recomandată în numeroase implementări OpenStack. Versiunea Epoxy aduce îmbunătățiri pentru erasure coding în Swift și stabilizează suportul „multi-attach” în Cinder. În ansamblu, platforma oferă cele mai multe opțiuni mature pentru stocare și integrare enterprise.

CloudStack 4.20.1 LTS organizează stocarea pe două niveluri: Primary Storage pentru volumele active ale mașinilor virtuale (discuri de sistem și de date), cu backend-uri precum NFS, iSCSI, Fibre Channel, Ceph RBD și spații logice pe cluster, și Secondary Storage pentru șabloane, imagini ISO și copii instantanee, de regulă pe NFS, cu posibilă integrare cu Amazon S3. Platforma nu include un serviciu nativ de obiecte similar Swift. Sunt disponibile alocarea subțire, snapshots și migrarea volumelor între clustere, însă nivelul de orchestrare și integrare este mai redus decât în OpenStack. Un avantaj îl constituie integrarea nativă a resurselor de stocare în modelul Zone → Pod → Cluster → Host, care permite control granular (de exemplu, profiluri de performanță diferite pe clustere), precum și mecanisme de marcare și supra-alocare pentru optimizarea utilizării.

OpenNebula 7.0 Phoenix adoptă o abordare pragmatică bazată pe conceptul de Datastore, local (LVM, ext4, ZFS) sau partajat (NFS, Ceph RBD, datastores VMware prin vCenter, dispozitive pe bloc partajate). Nu oferă un serviciu nativ de obiecte, dar integrează soluții externe (Ceph RGW, S3), mai ales pentru imagini de disc. Integrarea cu Ceph RBD este completă și utilizată atât pentru stocare persistentă, cât și pentru discuri ale mașinilor virtuale și discuri temporare, cu alocare dinamică și snapshots. Gestionarea volumelor se face printr-un sistem flexibil de drivere specifice fiecărui tip de Datastore, permițând alegerea backend-ului optim în funcție de sarcină. În scenariile VMware, platforma folosește datastores și VMFS via vCenter, iar administrarea stocării este realizată direct de serviciul oned și configurațiile sale. Funcționalitățile acoperă majoritatea scenariilor enterprise (alocare subțire, snapshots, clonare), însă cu un grad de complexitate și flexibilitate inferior OpenStack.

2.4.4. Conectivitatea și serviciile de rețea

OpenStack, în versiunea Epoxy 2025.1, oferă cel mai sofisticat mecanism de conectivitate prin componenta Neutron, care gestionează rețele virtuale și fizice, permite rețele izolate pe utilizator și susține topologii complexe de tip L2, L3, SDN și NFV. Segmentarea traficului este realizată prin marcatori VLAN (802.1Q), respectiv prin suprapuneri bazate pe VXLAN, GRE și GENEVE, ceea ce facilitează scenariile multiutilizator și implementări în cloud public. Neutron integrează interfețe de înaltă performanță cu SR-IOV, reducând traversarea stivei software a hipervizorului și îmbunătățind latențele. Sunt disponibile rutere virtuale, adrese IP flotante (floating IPs), servicii DHCP, DNS, VPN-as-a-Service și distribuirea încărcării (Load Balancing-as-a-Service). În mediile de categorie enterprise, integrarea cu soluții SDN precum Open vSwitch, OVN, OpenDaylight, Contrail sau Cisco permite control fin și granular al traficului și al politicilor de securitate. Versiunea Epoxy consolidează RBAC la nivel de Neutron și introduce optimizări pentru tunelarea cu GENEVE și pentru lanțuri de funcții de rețea în cadrul NFV. Complexitatea ridicată a configurațiilor și dependențelor cu alte module poate face din Neutron una dintre componentele cele mai provocatoare operațional.

CloudStack propune o arhitectură de rețea bine definită, dar mai simplă. Modelul Basic Networking asociază instanțele unui pool comun de adrese IP, adecvat implementărilor restrânse, iar Advanced Networking permite rețele izolate per utilizator folosind VLAN (802.1Q) sau VXLAN. În acest mod, componenta Virtual Router acționează ca dispozitiv virtual la nivel L3 și furnizează NAT, DHCP, firewall și port forwarding. Platforma poate integra distribuitoare de încărcare hardware sau software și oferă rutere virtuale redundante pentru disponibilitate ridicată. Integrarea cu platforme SDN, de exemplu VMware NSX sau Nicira NVP, este posibilă, însă mai puțin profundă și flexibilă decât în Neutron. Există suport pentru SR-IOV, cu integrare mai limitată, dependentă de hipervizor și de hardware. CloudStack nu include un controller SDN propriu și nu oferă un model NFV complet. Configurarea rămâne mai simplă și mai ușor de întreținut, cu o flexibilitate mai redusă în scenariile enterprise foarte complexe.

OpenNebula urmărește simplitatea și eficiența operațională. Virtual Networks pot utiliza izolarea prin VLAN (802.1Q), rețele suprapuse prin Open vSwitch sau alocări statice de game IP. Platforma integrează nativ Virtual Router, care asigură în regim software NAT, DHCP și distribuirea încărcării, permițând construirea de topologii controlate fără componente

suplimentare. În mediile enterprise, integrarea cu Open vSwitch permite scenarii overlay (de exemplu VXLAN), cu un nivel de complexitate mai redus decât în Neutron. OpenNebula acceptă plăci de rețea cu SR-IOV, expunând funcții virtuale direct către instanțe pentru cerințe de performanță ridicată. Nu există o arhitectură SDN sau NFV dedicată și nici integrare nativă cu orchestratoare SDN externe (de tip OpenDaylight sau ACI), accentul fiind pus pe control direct și simplitate, adecvate infrastructurilor cu complexitate de rețea moderată.

2.4.5. Planificatorul (scheduler) și alocarea resurselor

În OpenStack, versiunea Epoxy 2025.1, planificarea este una dintre cele mai sofisticate componente ale platformei. Planificatorul principal este integrat în Nova, iar funcționarea sa este extinsă prin serviciul Placement, care monitorizează precis resursele de procesare, memorie și disc, topologiile NUMA, asignările pe nuclee fizice (CPU pinning), acceleratoarele hardware (GPU, FPGA), precum și clase de resurse definite de administratori. Se utilizează un sistem de filtre și ponderi care aplică reguli pentru selectarea nodului optim, incluzând verificări ale resurselor disponibile, apartenența la grupuri de servere și respectarea regulilor de afinitate și anti-afinitate. Ponderile permit prioritizarea conform politicilor alese, de exemplu distribuirea uniformă a sarcinii sau preferințe legate de locație. Integrarea strânsă cu Placement API oferă vizibilitate asupra resurselor partajate (vGPU, SR-IOV, dispozitive de stocare pe blocuri). Platforma poate fi configurată pentru politici de disponibilitate ridicată, migrare în timp real, evacuare automată a instanțelor la defectarea unui nod și scenarii de plasare ce implică rezervări de resurse (reservation pools). Puterea acestui mecanism vine cu un cost operațional, necesitând efort susținut de optimizare și mentenanță.

CloudStack 4.20.1 LTS implementează un planificator mai simplu, dar robust, condus de politici de alocare și reguli la nivel de clustere și zone. Algoritmul identifică nodul care satisface cerințele de procesare, memorie, stocare și rețea ale instanțelor, folosind politici precum First Fit, Least Consumed sau User-Defined Weights. Sunt disponibile grupe de afinitate și anti-afinitate, migrare în timp real în hipervizoarele care o permit (KVM, VMware) și mecanisme de disponibilitate ridicată pentru mutarea automată a instanțelor. Integrarea cu ierarhia Zone → Pod → Cluster → Nod facilitează organizarea logică a resurselor. Platforma nu dispune de un serviciu separat echivalent Placement API și nu oferă aceeași granularitate pentru resurse speciale (vGPU, NUMA), însă nivelul de simplitate o face potrivită pentru scenarii medii și mari, cu flexibilitate mai redusă decât în OpenStack.

OpenNebula 7.0 Phoenix utilizează un planificator eficient, integrat direct în serviciul oned, care evaluează disponibilitatea nodurilor în funcție de procesoare, memorie, stocare, dispozitive PCI (inclusiv GPU), încărcarea nodului și etichete asignate resurselor. Începând cu această versiune, este disponibil Dynamic Resource Scheduling, permițând mutări proactive în funcție de sarcină, cu politici precum Packing, Striping sau expresii personalizate de rang (Custom Rank Expressions). Sistemul include reguli de afinitate și anti-afinitate, migrare în timp real în KVM și VMware și rezervări de resurse la nivel de cluster. Lipsa unui serviciu separat de tip Placement API conduce la o orchestrare mai simplificată a resurselor speciale (NUMA, vGPU), însă planificatorul este foarte performant și rapid, adecvat acolo unde latențele operaționale trebuie menținute la minimum, oferind opțiuni suficiente și pentru scenarii de categorie enterprise.

2.4.6. Managementul utilizatorilor și securitatea

OpenStack își bazează modelul de autentificare și autorizare pe Keystone, serviciu de identitate centralizat care susține mai multe mecanisme de autentificare: utilizator și parolă, token-uri (standard sau Fernet), directoare LDAP, Active Directory, SAML 2.0 și OpenID Connect (OIDC). Această flexibilitate permite integrarea în medii cu politici de Single Sign-On (SSO) și în organizații cu cerințe eterogene. Keystone gestionează token-uri cu valabilitate variabilă, oferă mecanisme de audit pentru identitate și acces și contribuie la trasabilitatea acțiunilor, logurile detaliate ale operațiilor fiind însă menținute de serviciile individuale. La nivelul API-urilor, mecanismul Role-Based Access Control (RBAC) a fost consolidat și este aplicat implicit prin politici explicite, ceea ce crește granularitatea și izolarea accesului. Modelul nativ de multi-tenancy permite izolarea proiectelor și utilizatorilor la nivel de rețea, stocare și

calcul, iar integrarea cu sisteme externe precum FreeIPA sau Active Directory simplifică sincronizarea conturilor și a grupurilor. Din perspectivă de securitate, sunt disponibile politici pentru gestionarea certificatelor, configurarea endpoint-urilor TLS, limitarea ratei apelurilor API și integrarea cu soluții de logging centralizat pentru detecția și analiza incidentelor.

CloudStack utilizează un model mai simplu de autentificare și autorizare, bazat pe domenii pentru organizarea conturilor. Implementarea multiutilizator este posibilă, cu o flexibilitate și granularitate mai reduse decât în OpenStack. Platforma oferă integrare LDAP, iar versiunile recente includ compatibilitate cu SAML pentru SSO, însă nu există suport nativ pentru OIDC. Controlul accesului bazat pe roluri permite roluri personalizate cu permisiuni precise, deși nivelul de detaliu al politicilor este inferior celui oferit de Keystone. Auditarea acțiunilor se realizează prin evenimente înregistrate în loguri, exportabile către soluții SIEM. Din punct de vedere al securității, sunt disponibile limitarea ratei apelurilor API, transport securizat prin TLS și protecții pentru consolele mașinilor virtuale. Gestionarea identității este integrată direct în platformă, fără un serviciu dedicat separat.

OpenNebula păstrează o abordare minimalistă, dar compatibilă cu cerințe de nivel enterprise. Modelul de utilizatori, grupuri și liste de acces și control (ACL) permite definirea de reguli precise asupra resurselor, cu integrare LDAP și Active Directory pentru sincronizarea utilizatorilor și grupurilor. Platforma susține autentificarea bazată pe token-uri și pe chei SSH, adecvată fluxurilor DevOps și accesului automatizat. Spre deosebire de OpenStack, nu implementează nativ SAML sau OIDC pentru autentificare unificată, deși extensii sau integrări personalizate pot oferi acest tip de funcționalitate. Începând cu versiunea 7.0, au fost extinse opțiunile RBAC pentru un control mai granular al permisiunilor. Sunt disponibile transport TLS, auditare și trasabilitate, cu un nivel de detaliu mai redus decât în OpenStack. Per ansamblu, gestionarea identității și a securității este echilibrată și adecvată infrastructurilor mici și medii, cu o complexitate operațională mai mică decât în soluțiile cu granularitate avansată.

2.4.7. Orchestrarea și automatizarea

OpenStack, în versiunile recente, oferă o soluție avansată de orchestrare și automatizare centrată pe Heat, serviciul care descrie infrastructura prin șabloane YAML și operationalizează principiile „Infrastructure-as-Code”. Prin Heat pot fi gestionate instanțe de calcul, volume de stocare, rețele virtuale, distribuitoare de încărcare, grupuri de auto-scalare și resurse personalizate, șabloanele având suport pentru parametri dinamici, ieșiri și condiții logice, ceea ce permite compoziții flexibile și reutilizabile. Integrarea mai strânsă cu mecanismele RBAC, precum și cu Placement API și Cyborg extinde orchestrarea către resurse hardware specializate, inclusiv acceleratoare GPU și FPGA. Platforma se integrează cu instrumente externe precum Ansible, Terraform, Puppet și Chef, beneficiind de furnizori oficiali și de contribuțiile comunității OpenStack. Mecanismele native de auto-scalare prin Heat Autoscaling Groups permit ajustarea dinamică a resurselor în funcție de încărcare. În mediile enterprise, nivelul de complexitate și sofisticare poate fi ridicat, presupunând un grad semnificativ de expertiză tehnică pentru implementare și operare eficiente.

CloudStack implementează un model mai simplificat de orchestrare. Platforma permite definirea de șabloane pentru mașinile virtuale, cu specificații privind procesoarele, memoria, discurile, conectivitatea de rețea și configurări aplicate după instanțiere, oferă suport pentru comenzi personalizate prin scripturi și integrează mecanisme precum cloud-init pentru automatizarea configurărilor interne. La nivel de orchestrare a infrastructurii în ansamblu nu există un serviciu dedicat echivalent cu Heat, însă este disponibilă o interfață API extensivă pentru administrarea programatică și integrarea cu Ansible, Terraform sau Jenkins, inclusiv furnizori oficiali Terraform pentru definirea infrastructurii ca și cod. Orchestrarea nativă rămâne mai limitată, fără motor intern de fluxuri de lucru și fără suport pentru orchestrarea resurselor hardware specializate, precum acceleratoarele GPU sau FPGA. Pentru scenarii enterprise de complexitate medie, capabilitățile API sunt, de regulă, suficiente, însă pentru implementări complexe, cum ar fi infrastructurile NFV sau orchestrările cu resurse multiple la scară largă, platforma este mai puțin potrivită decât OpenStack.

OpenNebula implementează un model de orchestrare simplu, dar eficient, centrat pe OneFlow, care permite definirea de Service Templates și orchestrarea aplicațiilor cu mai multe mașini virtuale, cu relații de dependență, politici de scalare și acțiuni personalizate. OneFlow

admite roluri cu politici proprii de scalare și instanțiere, iar User Contextualization permite injectarea de scripturi și variabile personalizate pentru configurări post-instanțiere. Pentru Infrastructure-as-Code, există integrare cu Terraform printr-un furnizor oficial și suport pentru hooks și drivere personalizate. Platforma nu dispune de un orchestrator generalist precum Heat, iar orchestrarea resurselor speciale este mai simplificată. Pentru un context de utilizare mediu, opțiunile oferite sunt suficiente, fără complexitatea și flexibilitatea avansată asociate modulului Heat din OpenStack.

2.4.8. Telemetria și monitorizarea

În versiunile recente, OpenStack integrează un ansamblu coerent de servicii pentru monitorizare și telemetrie, cu Ceilometer drept componentă principală de colectare a datelor privind utilizarea resurselor, inclusiv instanțe de calcul, volume de stocare, rețele, utilizatori și proiecte. Ceilometer operează atât în mod de interogare periodică, cât și în mod declanșat de evenimente, în funcție de capabilitățile fiecărui serviciu monitorizat. Arhitectura s-a orientat către utilizarea Gnocchi ca backend pentru metrici, datorită modelului pe serii de timp care susține eficient volume mari de date. Panko a fost utilizat pentru stocarea evenimentelor și analiză scalabilă, deși dezvoltarea sa este în prezent suspendată. Integrarea dintre servicii s-a consolidat, iar Ceilometer permite exportul direct al metricilor către sisteme externe, de exemplu Prometheus, Elasticsearch sau Kafka, facilitând integrarea în infrastructuri enterprise. Mecanismele de alarmare sunt asigurate de Aodh, care poate reacționa la metricele colectate de Ceilometer, respectiv la evenimentele stocate în Panko, inclusiv pentru declanșarea de acțiuni automate, cum ar fi operații de auto-scalare. Politicile RBAC se extind și asupra serviciilor de telemetrie, ceea ce securizează accesul la metrici și evenimente. Prin această arhitectură și prin suportul nativ pentru integrare cu instrumente externe, OpenStack se poziționează ca soluție de referință în medii enterprise unde auditarea și telemetria la scară mare sunt esențiale.

CloudStack implementează un model mai simplu, colectând date despre CPU, memorie, stocare și trafic de rețea prin Management Server și agenții hipervizoarelor, cu persistență în baza de date și expunere prin API și interfața grafică. Platforma nu dispune de o arhitectură dedicată exclusiv telemetriei, similară Ceilometer. Pentru scenarii enterprise este frecventă integrarea cu Zabbix, Prometheus sau Grafana, statisticile putând fi extrase prin API. Nu există un mecanism nativ de stocare a evenimentelor, comparabil cu Panko, și nici un motor integrat pentru managementul alarmelor, astfel că alertarea și corelarea se realizează în instrumente externe. Telemetria rămâne funcțională și stabilă, dar semnificativ mai simplă decât în OpenStack.

OpenNebula adoptă o abordare minimalistă. Metricile privind nodurile de găzduire, mașinile virtuale, stocarea și rețeaua sunt colectate de serviciul principal, oned, și de driverele de monitorizare, cu stocare în baza de date și acces prin Sunstone și API. Platforma nu include componente separate pentru telemetrie sau stocarea evenimentelor, însă permite integrarea cu instrumente externe. În versiunea 7.0 este disponibil suport îmbunătățit pentru Prometheus, prin exportul metricilor în format compatibil, facilitând utilizarea în ecosisteme de observabilitate precum Grafana. Există hooks la evenimentele ciclului de viață al VM-urilor, capabile să execute scripturi externe. Nu este prezent un serviciu nativ de tip Aodh și nici o bază de date proprie pentru serii de timp. Pentru implementări mici și medii, această simplitate este de regulă suficientă și reduce semnificativ povara operațională.

2.4.9. Interfața de management

OpenStack include Horizon, consolă web extensibilă pentru majoritatea serviciilor, prin care se gestionează instanțe, rețele, volume, imagini, utilizatori, proiecte și politici de securitate. Pot fi adăugate panouri personalizate pentru Heat, Ceilometer, Mistral, Cyborg. Limitările semnalate frecvent privesc complexitatea și timpii de răspuns în implementări de mari dimensiuni. Pentru îmbunătățirea experienței, în Epoxy este dezvoltat Skyline, interfață web modernă construită pe React și API REST complet asincron, concepută să înlocuiască parțial Horizon. Skyline oferă viteză sporită și adaptare la dispozitive portabile, fără a acoperi încă toate funcționalitățile Horizon. Complementar, OpenStack CLI furnizează un client unificat

pentru linia de comandă, folosit intensiv în scenarii DevOps și scripturi personalizate. Platforma expune API-uri REST standardizate și bine documentate, cu SDK-uri oficiale în Python, Go, Java, C# și altele, ceea ce facilitează automatizarea și integrarea în CI/CD.

CloudStack oferă CloudStack UI, interfață web clar structurată, construită inițial pe AngularJS și migrată treptat spre Vue.js în actualizările recente. Interfața permite administrarea completă a infrastructurii, incluzând mașini virtuale, volume, rețele, șabloane, utilizatori, conturi și domenii, fiind apreciată pentru organizarea vizuală intuitivă și ergonomică, precum și pentru vizualizările detaliate ale ierarhiei infrastructurii, utile în medii complexe. Din perspectiva automatizării, CloudMonkey CLI permite execuția rapidă a comenzilor prin API, cu autocompletare, integrare în scripturi și acoperire completă a entităților gestionate de platformă. API-ul REST este complet, bine documentat și stabil între versiuni, permițând integrarea cu Terraform, Ansible, Jenkins.

OpenNebula integrează Sunstone, interfață web simplă și intuitivă, aliniată filozofiei minimaliste a platformei. Sunstone asigură administrarea completă a infrastructurii, incluzând mașini virtuale, șabloane, rețele virtuale, stocare, utilizatori și permisiuni. Interfața este rapidă, cu consum redus de resurse, și se poate extinde prin plugin-uri externe, fără a necesita o arhitectură complexă sau multiple straturi de backend. În versiunea 7.0 au fost aduse îmbunătățiri de UI și UX, inclusiv un mod de filtrare a interfeței în funcție de nivelul utilizatorului, cu vizualizare simplificată pentru utilizatori obișnuiți și una avansată pentru administratori. Pentru automatizare, oned CLI permite administrarea integrală din linia de comandă, cu suport pentru scripturi complexe și execuție rapidă în medii DevOps. Platforma oferă API REST și API XML-RPC, ambele bine documentate și compatibile cu majoritatea limbajelor de programare, deși ecosistemul de SDK-uri externe este mai restrâns decât în OpenStack. În mediile enterprise, combinația Sunstone și CLI acoperă majoritatea cerințelor de operare și integrare.

2.4.10. Edge computing și cloud hibrid

OpenStack a avansat consistent în zona edge computing, beneficiind de arhitectura modulară și de proiecte conexe. StarlingX, proiect open-source bazat pe OpenStack, este optimizat pentru scenarii edge și telecom, combină determinismul specific sistemelor de operare real-time, gestionarea eficientă a resurselor hardware limitate și instanțierea rapidă în locații distribuite. Integrează OpenStack cu Kubernetes pentru orchestrarea serviciilor la marginea rețelei, fiind adoptat în telecomunicații, transport și industria auto. Ecosistemul susține și arhitecturi edge lite, în care planul de control rămâne centralizat, iar nodurile de găzduire sunt distribuite geografic. În cloud hibrid, OpenStack se integrează cu furnizori publici, de exemplu AWS, Azure, Google Cloud Platform, prin plugin-uri Neutron SDN sau soluții externe de orchestrare. Distribuțiile comerciale includ componente dedicate, de exemplu integrarea Red Hat OpenShift Container Storage cu Cinder și Swift, care facilitează migrarea sarcinilor între mediile locale și cloud public. Epoxy a îmbunătățit integrarea cu Kubernetes, inclusiv prin serviciul Magnum, consolidând strategii de cloud hibrid și multi cloud.

CloudStack are un suport limitat pentru edge computing în sens strict, fiind conceput în principal pentru centre de date tradiționale cu infrastructură centralizată. Prin modelul pe zone, pod uri și clustere, poate fi implementat în topologii distribuite geografic, fiecare zonă reprezentând un centru de date distinct, managementul rămâne centralizat, iar sarcinile pot fi distribuite în funcție de locația utilizatorilor. Platforma nu include componente dedicate pentru procesare în timp real, pentru medii cu constrângeri hardware sau pentru sarcini cu latență scăzută, caracteristice edge computing ului modern. În cloud hibrid, integrarea este mai degrabă limitată, interacțiunea cu cloud uri publice se realizează prin API și instrumente externe, fără proiecte oficiale pentru multi cloud. Migrarea sarcinilor către cloud public se bazează pe Terraform sau scripturi personalizate, în absența unui orchestrator nativ pentru scenarii hibride. În practică, CloudStack rămâne mai potrivit pentru infrastructuri locale tradiționale decât pentru edge computing sau orchestrare multi cloud.

OpenNebula a progresat semnificativ în edge computing, trecând de la orientarea exclusivă către centre de date la administrarea distribuțiilor geografice complexe. Conceptul de Edge Clusters permite management centralizat pentru resurse aflate la distanță, implementabile atât on premises, cât și în infrastructuri publice, inclusiv AWS și Azure, datorită

suportului integrat pentru cloud uri publice. Platforma permite instanțiere directă în cloud public printr-o interfață unică, facilitând implementări multi-cloud. Integrarea cu containere LXD se potrivește scenariilor edge, unde se urmărește amprentă redusă și consum minim de resurse. În cloud hibrid, OpenNebula oferă integrare directă cu AWS și Azure, cu suport pentru optimizarea costurilor și alocare dinamică, orchestrând simultan resurse locale și publice pe baza acelorași șabloane și politici. Nu există proiecte dedicate de complexitatea StarlingX, însă simplitatea arhitecturală și integrarea funcționalităților răspund bine cerințelor infrastructurilor mici și medii.

2.4.11. Maturitate, dezvoltare și comunitate

OpenStack este una dintre cele mai mature și extinse platforme open-source pentru cloud computing la scară mare în mediile profesionale. Lansat în 2010, proiectul a cunoscut o expansiune semnificativă și este utilizat în sute de organizații mari, de la operatori telecom la instituții academice și organizații guvernamentale, inclusiv CERN. Comunitatea este coordonată de OpenInfra Foundation, cu mii de contributori și numeroase companii membre (Red Hat, Canonical, Mirantis, Huawei, HPE și altele). La fiecare ciclu de lansare, în prezent la șase luni, platforma însumează mii de modificări de cod și parcurge un proces riguros de verificare, asigurând stabilitatea și corectarea rapidă a vulnerabilităților. Ecosistemul comercial include distribuții orientate către mediul profesional, cu asistență tehnică, SLA-uri și integrare cu instrumente comerciale, iar documentația este amplă și bine structurată, susținută de o prezență constantă la evenimente precum OpenInfra Summit. Complexitatea ridicată rămâne, totuși, o barieră pentru unele organizații mici și medii. În ultima versiune, maturitatea a fost consolidată prin stabilizarea unor proiecte conexe precum Cyborg și Skyline, însă instalarea și operarea eficientă continuă să necesite un nivel înalt de expertiză tehnică.

CloudStack, ajuns la versiunea 4.20.1 LTS, este de asemenea o platformă matură, lansată în 2010 și donată ulterior Apache Software Foundation, unde este proiect de nivel superior, cu guvernanta deschisă, reguli stricte de verificare și cicluri de lansare stabile. Comunitatea, mai mică decât cea a OpenStack, este activă și stabilă, cu contribuții notabile din partea companiilor ShapeBlue și StorPool. Platforma se remarcă prin stabilitate și predictibilitate: API-ul a rămas în mare parte compatibil între versiunile majore, ceea ce favorizează actualizări fără întreruperi prelungite. Este considerabil mai simplă de instalat și operat decât OpenStack, cu consum redus de resurse și cerințe hardware mai modeste. Ecosistemul comercial este însă mai restrâns, iar documentația, deși clară, nu atinge amploarea celei din OpenStack. Rezultă o opțiune matură pentru organizațiile care urmăresc o soluție stabilă fără complexitatea operațională specifică OpenStack.

OpenNebula, lansată în 2008, a evoluat semnificativ, fiind susținută de OpenNebula Foundation și de o comunitate mai mică decât cele ale OpenStack și CloudStack. Platforma este preferată de organizațiile care caută simplitate, implementare rapidă și complexitate operațională redusă. Dezvoltată și întreținută de un nucleu mai restrâns de contributori, este sprijinită în principal de OpenNebula Systems, care oferă asistență profesională și servicii comerciale. Un avantaj major îl reprezintă maturitatea în scenarii pentru organizații mici și medii, precum și în edge computing, unde contează simplitatea și consumul redus de resurse. Deși ecosistemul comercial este mai limitat, platforma a atras o adopție notabilă în cercetare, educație și companii care evită complexitatea ecosistemului OpenStack. Marketplace-ul OpenNebula oferă appliance-uri și șabloane gata de utilizat. Totuși, nu dispune de aceeași amploare comunitară și comercială ca OpenStack, rămânând o opțiune matură pentru medii mai simple și scenarii de edge, fără nivelul de complexitate și integrare cerut de organizațiile mari.

2.4.12. Concluzii privind comparația platformelor de virtualizare

Analiza comparativă dintre OpenStack, CloudStack și OpenNebula evidențiază diferențele tehnice relevante și conturează scenariile în care fiecare platformă este avantajoasă, fără generalizări. Rezultă un tablou util pentru organizațiile și instituțiile interesate de o soluție cloud open-source adaptată nevoilor proprii, fie pentru implementări de mari

dimensiuni, fie pentru infrastructuri locale mai simple sau pentru scenarii moderne de edge computing.

OpenStack se remarcă prin complexitate și amplitudine funcțională, fiind organizat în servicii specializate pentru calcul, rețea și stocare. Versatilitatea îi recomandă utilizarea în medii extinse, cu cerințe avansate de integrare, securitate și scalabilitate, însă presupune efort semnificativ de implementare și operare, expertiză tehnică solidă și resurse consistente. Pe plan comercial, beneficiază de oferte susținute de furnizori și de o comunitate activă, cu evoluții și în direcția aplicațiilor containerizate și a infrastructurilor edge.

CloudStack privilegiază robustețea și simplitatea operațională. Modelul său mai compact permite punerea în funcțiune rapidă și administrarea facilă, iar organizarea în zone, poduri și clustere aduce claritate în proiectarea și managementul resurselor. Stabilitatea API-urilor între versiuni simplifică actualizările și menține compatibilitatea integrărilor. Limitările apar în scenarii complexe de cloud hibrid sau edge, unde lipsește un mecanism nativ de orchestrare, utilizatorii apelând frecvent la instrumente externe. Ecosistemul comercial este mai restrâns decât în cazul OpenStack, deși comunitatea rămâne dinamică.

OpenNebula pune accent pe ușurința în utilizare și pe eficiența resurselor. Evoluția din versiunea 7.0 Phoenix introduce Edge Clusters pentru extinderea către locații distante sub management centralizat. Integrarea directă cu AWS și Azure deschide scenarii hibride, deși nivelul de orchestrare între medii cloud este mai puțin sofisticat decât în soluții specializate. Platforma este apreciată pentru implementarea rapidă, cerințele reduse de întreținere și interfața Sunstone, orientată spre simplitate și eficiență, în contextul unei comunități și al unui sprijin comercial mai temperate.

Alegerea depinde de contextul concret. OpenStack se potrivește organizațiilor mari, cu nevoi avansate și integrare complexă. CloudStack oferă stabilitate și predictibilitate fără nivelul de complexitate al OpenStack. OpenNebula răspunde cerințelor de viteză, eficiență și simplitate, în special pentru medii mici și medii sau pentru scenarii edge. În raport cu arhitectura experimentală analizată în această lucrare, OpenNebula se conturează drept opțiunea adecvată, argumentele detaliate fiind prezentate în capitolul următor.

Capitolul III - Propunere de arhitectură experimentală pentru determinarea tiparelor de intrare/ieșire

3.1. Motivația testelor experimentale

Evaluarea performanței sistemelor de virtualizare care rulează simultan multiple mașini virtuale necesită caracterizarea tiparelor de acces la disc, dincolo de volumul total transferat: distribuția temporală a cererilor, dimensiunea blocurilor, numărul de operații și raportul secvențial/aleatoriu sunt determinante. Debitul de date, definit ca volumul total transferat într-un interval, rămâne parametrul fundamental, iar un algoritm inteligent de plasare a datelor poate crește lățimea de bandă totală cu aproximativ 20% față de o localizare aleatorie, în special pentru sarcini mixte [130]. Tiparele de acces influențează decisiv performanța: secvențialitatea citirilor/scrierilor poate îmbunătăți subsistemul de stocare, iar diferențele dintre accesul aleatoriu și cel secvențial conduc la variații notabile ale lățimii de bandă și ale numărului de operații de intrare/ieșire [131]. Raportul scrieri/citiri depinde de tipul serverului, aplicații, sistem de operare și sistem de fișiere: de pildă, un server Microsoft Exchange proiectat pentru 1:1 prezintă în practică aproximativ 2:1 în favoarea scrierilor [132].

Analizele asupra serverelor de compilare și distribuție (WBS, DTRS) indică un trafic mai echilibrat, dar dominat de cereri mici (4-8 KB), cu acces relativ aleatoriu; scrierile cresc în etapele de generare a fișierelor și de completare a jurnalelor. În mediile cloud comerciale, pe clustere Hadoop, compararea stocării locale NVMe în mașini virtuale cu stocarea pe blocuri în rețea arată o reducere de circa 7 ori a latenței (de la 8200 μs la 1200 μs) în favoarea NVMe, cu impact major asupra aplicațiilor interactive sau OLTP; testele au folosit fio cu tipar „rw50” (50% scrieri/50% citiri) [133], [134]. O arhitectură extinsă de analiză a performanței I/O în medii virtualizate (SPA) combină regresie statistică și rețele Petri (QPN), calibrate pe rulari locale ale Filebench sau FFSB în interiorul mașinilor virtuale; deși utilă, abordarea este invazivă, nu corelează evenimentele de orchestrare (deploy, boot) cu performanța observată și nu a mai fost actualizată după 2015 [135], [136].

Trasări din două mari servicii de stocare pe bloc (Alibaba Cloud, Tencent Cloud) indică schimbări substanțiale față de datele tradiționale (MSRC): volumele sunt predominant orientate spre scrieri (~90% dintre volume sunt dominate de scrieri), iar peste 75% dintre cereri au dimensiuni sub 32 KB [8]. Din perspectiva clientului, pe Amazon S3 (instanțe EC2), mărimea cererilor și paralelismul pot crește semnificativ lățimea de bandă; performanța clientului devine limitativă pentru cereri mici, distanța geografică afectează mai ales latența, iar interferențele între cereri mixte pot degrada drastic performanța [20].

În domeniul NFS, literatura evidențiază diversitatea metodologiilor și variabilitatea tiparelor. Instrumente open-source precum nfsdump (captură/decodare NFSv2/v3) și nfsscan (agregare/tabele) separă colectarea de analiză și facilitează organizarea pe utilizator, grup, client sau fișier [138]. Înregistrări reprezentative dintr-o infrastructură de email universitară arată activitate preponderent de citire cu fișiere active bine delimitate, în timp ce un mediu de cercetare informatică prezintă un raport scriere/citire mai ridicat și o pondere mare a operațiilor pe metadata [142]. Medii comerciale de mare volum, precum un studio de animație (~2,4 miliarde de operații NFS/zi), au necesitat optimizări software pentru capturi fără pierderi [139]. Micro-benchmark-uri NFSv4.1 arată că prefetching-ul, readahead-ul și sesiunile/delegările pot varia semnificativ performanța în funcție de aplicație, client și parametrii rețelei [140]. Instrumente mai noi, inclusiv soluții dedicate de mare capacitate, oferă granularitate superioară analizei traficului [141].

Rezultă că tiparele sunt profund dependente de context; optimizarea necesită monitorizare și analiză în mediul propriu. În consecință, se propune o arhitectură cu unelte pentru determinarea tiparelor de acces I/O într-un sistem de stocare partajat al unei platforme de virtualizare și o metodologie de evaluare a soluțiilor de stocare după adaptabilitatea la sarcinile reale.

3.2. Arhitectură propusă pentru testări

Obiectivul lucrării este identificarea unui tipar de acces la sistemul de stocare într-un mediu de virtualizare destinat gestionării și execuției mașinilor virtuale. Așa cum s-a arătat anterior, modelarea teoretică a acestui tipar este dificilă, în special în contextul utilizării platformei în scopuri educaționale. Diversitatea aplicațiilor rulate în mașinile virtuale și varietatea destinațiilor lor de utilizare produc o variabilitate ridicată a comportamentului de acces la resursele de stocare, ceea ce face improbabilă formularea unei prognoze riguroase și general valabile.

3.2.1. Arhitectura logică

Pentru determinarea tiparului de acces la stocare al mașinilor virtuale a fost adoptată o abordare empirică, bazată pe măsurarea directă a operațiunilor de intrare ieșire în plan cantitativ și calitativ. Arhitectura experimentală din figură are rolul de a colecta pe toată durata testelor informații detaliate privind lățimea de bandă utilizată, volumul de operații și latențele asociate accesului la sistemele de stocare. Datele sunt ulterior corelate temporal, analizate prin reprezentări grafice ale activităților observate și folosite la extragerea parametrilor relevanți, în vederea conturării unui model reprezentativ pentru acest tip de mediu virtualizat.

Monitorizarea vizează două puncte principale ale sistemului de stocare, la nivelul hipervizoarelor, unde se colectează informații agregate despre operațiile generate de mașinile virtuale, și la nivelul serverului de stocare, unde se urmăresc citirile și scrierile așa cum sunt percepute direct de sistemul de stocare.

Arhitectura propusă include următoarele componente, descrise funcțional în secțiunile următoare ale lucrării:

1. Server de control și monitorizare a evenimentelor, cu rol de orchestrare a inițializării instanțelor de mașini virtuale și de supraveghere a ciclului lor de viață. Sistemul înregistrează și salvează toate stările parcurse de fiecare instanță până la terminarea execuției.
2. Sistem de agregare a datelor și generare de grafice, responsabil de colectarea fișierelor de monitorizare de la nodurile de calcul, sincronizarea temporală a datelor și producerea graficelor detaliate pentru fiecare rulare experimentală, precum și de calculul valorilor medii pentru toți indicatorii observați.
3. Modul de colectare la nivelul hipervizorului, pornit pe fiecare nod de calcul care găzduiește mașini virtuale, pentru înregistrarea activității de acces la discurile virtuale și agregarea datelor în vederea analizării ulterioare.
4. Modul de colectare la nivelul sistemului de stocare, destinat supravegherii și centralizării datelor privind activitatea agregată a sistemului de stocare, cu monitorizarea accesărilor generate de instanțele virtuale din nodurile de calcul.
5. Sistem de monitorizare a parametrilor de funcționare a serverelor, care urmărește resursele critice ale infrastructurii cloud, inclusiv gradul de utilizare a procesorului, încărcarea memoriei principale și nivelul general de încărcare al sistemului de operare.

3.2.2. Arhitectura hardware

Pentru determinarea tiparului de acces la stocare a fost proiectată și implementată o infrastructură hardware dedicată, organizată ca un mediu cloud distribuit. Componentele fizice sunt configurate pentru a asigura colectarea și corelarea datelor relevante (lățime de bandă, volum de operații, latențe) pe întreaga durată a testelor, cu integrare ulterioară în reprezentări grafice și extragerea parametrilor necesari modelării comportamentului de intrare ieșire. Monitorizarea vizează două puncte principale: la nivelul hipervizoarelor, unde se colectează operațiile agregate ale mașinilor virtuale, respectiv la nivelul sistemului de stocare, unde se observă direct citirile și scrierile percepute de subsistemul de stocare.

Infrastructura include următoarele sisteme și entități:

- un server principal pentru stocare;
- opt noduri de calcul pentru rularea și găzduirea mașinilor virtuale;
- un server pentru orchestrarea cloud-ului;

- un server dedicat managementului și controlului sistemului de testare;
- un server pentru depozitul de fișiere;
- un segment de rețea dedicat traficului aferent discurilor mașinilor virtuale (SAN);
- un segment de rețea dedicat traficului mașinilor virtuale;
- un segment de rețea dedicat managementului hipervizoarelor și controlului testelor.

Caracteristicile hardware ale serverului de monitorizare și control și ale celui pentru depozitul de fișiere nu influențează semnificativ performanțele măsurate și nu sunt detaliate. Serverul desemnat ca „server de stocare” nu este un echipament de stocare dedicat în sens strict, ci a fost ales pentru această funcție în cadrul testelor deoarece dispune de cea mai mare capacitate RAM din infrastructura disponibilă. Alegerea urmărește exploatarea memoriei volatile pentru simularea unui spațiu de stocare de înaltă performanță, rolul său fiind temporar și funcțional, cu justificări prezentate într-o secțiune viitoare.

Conectivitatea este asigurată printr-un switch Supermicro SSE-F3548S(R) cu 48 de porturi de 10, 25 Gbps și 6 porturi de 40, 50, 100 Gbps, configurat pentru izolarea strictă a segmentelor de rețea. Rutarea între segmente este realizată de echipamente externe adiacente infrastructurii de testare, iar switchul operează exclusiv la nivelul 2 al rețelei, pentru latențe minime și comportament predictibil în sarcină. Pentru segmentul dedicat stocării, fiecare server are interfețe de rețea separate, alocate exclusiv fluxurilor de date aferente accesului la stocare, eliminând competiția cu traficul de control și de management și asigurând un mediu izolat, cu performanțe constante.

Arhitectura include și un server de fișiere cu funcționalitate dublă. Pe de o parte, acesta rulează un serviciu web (apache) care furnizează pachetele pentru platforme Windows. Pe de altă parte, găzduiește un depozit local complet pentru Ubuntu 24.04 LTS, sursă principală de instalări și actualizări pentru mașinile virtuale Linux. Mașinile virtuale și acest server sunt conectate într-o rețea dedicată, folosind interfețe separate pe serverele de găzduire. Segmentul pentru managementul nodurilor de găzduire și controlul testelor este, de asemenea, izolat la nivelul 2.

Aceste opțiuni metodologice elimină variabile externe care ar putea afecta consistența rezultatelor. În particular, se evită dependența de lățimea de bandă a conexiunii la Internet și întârzierile datorate accesului la surse externe de pachete, obținându-se un mediu stabil, reproductibil și lipsit de factori perturbatori externi.

Caracteristicile tehnice ale sistemelor server folosite în cadrul experimentului sunt descrise într-un tabel în lucrare.

3.2.3. Componentele arhitecturii pentru realizarea experimentelor

3.2.3.1. Ciclurile de viață ale mașinilor virtuale

Ipoteza de proiectare a arhitecturii experimentale este că evaluarea realistă a activității de acces la stocare impune ca mașinile virtuale să execute sarcini efective, în special operații cu interacțiune intensivă cu sistemul de fișiere. În acest scop, în interiorul fiecărei instanțe au fost integrate scripturi care declanșează automat procese de instalare și actualizare a pachetelor software. Alegerea acestor activități reflectă comportamente tipice în medii educaționale, unde utilizatorii recurg frecvent la instalarea de aplicații și la personalizarea mediului de lucru. Acest tip de sarcini generează un volum semnificativ de operații de citire și scriere și permite observarea, în condiții apropiate de realitate, a performanței subsistemului de stocare.

Pentru evaluarea performanței și interpretarea corectă a rezultatelor, monitorizarea precisă a ciclului de viață al instanțelor este esențială. În scenarii de orchestrare automată, cu inițializări paralele ale unui număr mare de instanțe, este necesar un model coerent de urmărire a stărilor, de la inițializare până la finalizarea operațiilor planificate. Definirea unui set standardizat de stări intermediare permite corelarea etapelor de execuție cu evenimentele relevante din infrastructură și traficul de rețea și, totodată, măsurarea duratelor asociate fiecărei faze. Prin marcarea unor repere temporale clare, precum momentele de inițializare, alocare, lansare, disponibilitate a sistemului de operare și execuție efectivă a sarcinilor, se obține o imagine detaliată asupra performanței ciclului complet de viață al unei instanțe.

Pentru identificarea clară a etapelor prin care trec mașinile virtuale pe parcursul inițializării și execuției, au fost stabilite și formalizate următoarele stări caracteristice:

- **instantiate:** momentul în care este programată crearea mașinii virtuale în sistemul de virtualizare, fiind injectate informațiile necesare pentru intrarea în ciclul de programare pentru execuție (scheduler). Evenimentul este raportat de scriptul care pornește mașina virtuală;
- **deploy:** momentul în care începe crearea propriu-zisă a mașinii virtuale, prin generarea structurilor XML care o definesc și prin inițierea creării imaginii de disc pe sistemul de stocare, de regulă prin copiere, dar uneori prin crearea de linkuri simbolice;
- **boot:** momentul în care mașina virtuală este lansată în execuție pe nodul ales de planificator. Starea este raportată de platforma de virtualizare printr-un mecanism intern declanșat automat;
- **ready:** momentul în care sistemul de operare din mașina virtuală a fost încărcat și este pregătit să preia comenzi. Starea este raportată de scripturile lansate automat imediat după pornirea sistemului de operare;
- **start:** momentul în care încep operațiile de descărcare și instalare a pachetelor software în interiorul mașinii virtuale, raportat de scripturile de inițializare;
- **end:** momentul în care au fost finalizate operațiile de descărcare, actualizare și instalare, raportat de scripturile automate.

3.2.3.2. Descrierea componentelor și a serverului pentru controlul și monitorizarea evenimentelor

Elementul central al arhitecturii logice pentru realizarea experimentelor este serverul de control și monitorizare a evenimentelor alcătuit din două componente care funcționează în tandem.

Prima componentă este un script de orchestrare, proiectat să gestioneze întregul flux de testare și care integrează următoarele funcționalități:

- interpretează argumentele din linia de comandă, sub forma unei liste de perechi `<model_vm_1>:<număr_instanțe_1>,<model_vm_2>:<număr_instanțe_2>` separate prin spațiu; fiecare pereche indică modelul de mașină virtuală ce urmează a fi instanțiat și numărul de instanțe aferent; pot fi specificate două tipuri de mașini virtuale care rulează concomitent, iar pentru testarea unui singur model se pune valoarea 0 pentru al doilea;
- realizează sincronizarea ceasurilor pe toate nodurile de găzduire și serverele adiacente participante, prin rețea, folosind un server public de timp, pentru a asigura consistența temporală a măsurărilor;
- inițializează execuția componentei secundare - un server web dedicat monitorizării evenimentelor ciclului de viață al VM-urilor; procesul este lansat în paralel, i se memorează PID-ul pentru oprire la momentul potrivit;
- creează, prin platforma de virtualizare, instanțele corespunzătoare primului model și înregistrează identificatorii acestora, marcând în serverul web starea „instantiated” pentru fiecare VM (modul concret diferă în funcție de tipul de încărcare simulat, detaliat într-o secțiune viitoare);
- creează în mod analog instanțele celui de-al doilea model, salvează identificatorii și marchează starea „instantiated”;
- intră într-o buclă de monitorizare periodică, interogând serverul web pentru a verifica atingerea stării finale („end”) de către toate instanțele active;
- în aceeași buclă, identifică instanțele eșuate („failed”) și recrează automat înlocuitori;
- la încheierea ciclului, generează un fișier de raport ce conține configurația completă a testului și marcasele temporale ale momentelor de început și sfârșit;
- oprește serverul web de monitorizare a evenimentelor VM;
- oprește toate instanțele lansate, eliberând resursele;
- așteaptă eliberarea memoriei ocupate de imaginile VM; chiar dacă ștergerea este comandată, sistemul are nevoie de timp pentru eliberarea completă a spațiului din discul virtual creat în RAM, mai ales pentru imagini de dimensiuni semnificative;

- dacă linia de comandă conține seturi suplimentare de parametri pentru alte teste, reia procesul în mod automat începând cu sincronizarea temporală.

A doua componentă a serverului este un server web care colectează și corelează evenimentele temporale transmise de VM-uri pe durata ciclului de viață. Sunt urmărite șase evenimente: „instantiated”, „deploy”, „boot”, „ready”, „start”, „end”, care descriu tranzițiile fundamentale, de la inițializare la terminare. Serverul HTTP expune două endpoint-uri principale: /status pentru vizualizarea în timp real a stării VM-urilor și /go-status pentru coordonarea sincronizată a experimentelor. Mecanismul de coordonare se bazează pe un semafor: când toate VM-urile ating starea „ready”, semaforul este setat la „1” (pregătirea pentru începerea experimentului); după finalizarea tuturor scripturilor, semaforul revine la „0” (încheierea ciclului). Colectarea datelor de performanță se face prin înregistrarea timpilor absoluți ai evenimentelor și calculul duratelor dintre evenimente consecutive, cu precizie de milisecunde; la final, pentru fiecare instanță se salvează în format CSV toate informațiile ciclului de viață, iar interfața web permite monitorizarea în timp real.

Al doilea element din arhitectura propusă este sistemul de agregare a datelor și generare de grafice, executat pe o stație separată. Acesta colectează automat fișierele generate de modulele de monitorizare distribuite (utilizare CPU, acces la disc, lățime de bandă, latențe), le transferă pe un server central și le stochează într-o structură ierarhică ce reflectă fiecare rulare (moment, configurație, parametri). Ulterior, scripturi Python procesează și reprezintă grafic datele (de ex. evoluția ratei de transfer, distribuția VM-urilor pe noduri, corelația utilizare CPU-timpul testului). În paralel, sunt analizate momentele temporale ale ciclului de viață (minim/medie/maxim), salvate în CSV și folosite de un modul care evaluează mediile pe intervale definite între evenimente, urmărind variații de lățime de bandă, număr I/O și latențe. Un ultim modul agregă global rezultatele în statistici și vizualizări comparative, utile validării ipotezelor și formulării concluziilor.

Elementul nr. 3 este modulul de colectare la nivelul hipervizoarelor. Pornit pe fiecare nod de calcul, acesta înregistrează activitatea de acces la discurile virtuale și agregă datele pentru o analiză coerentă a comportamentului subsistemului de stocare. Monitorizarea are rezoluție de o secundă și utilizează interfața libvirt [143]. Scriptul (Python, cu API-ul libvirt) enumeră VM-urile în execuție și colectează pentru fiecare statisticile dispozitivului de bloc virtual (număr cereri citire/scriere, volume citite/scrise, operații de „flush” unde este disponibil). Prin blockStatsFlags se încearcă extragerea timpilor totali de procesare ai citirilor/scririlor (în nanosecunde), din care se calculează latențe medii (convertite în microsecunde). Datele sunt apoi agregate la nivel de nod (total cereri, volum transferat pentru lățime de bandă cvasi-instantanee, latențe medii).

Elementul al patrulea este modulul de colectare la nivelul sistemului de stocare, care monitorizează I/O pe serverul ce partajează spațiul de stocare către nodurile de găzduire. Obiectivul este surprinderea fidelă a comportamentului agregat al accesului concurent, asincron și variabil. Monitorizarea vizează volumul de date, numărul operațiilor, tipul (citire/scriere) și, unde este posibil, latențele. La nivel de sistem Linux, se utilizează interfețe precum /proc/diskstats, care furnizează statistici detaliate pentru fiecare dispozitiv de bloc (număr de operații, cantități transferate, durate). În context multi-client (de ex. NFS), multe operații ajung ca apeluri de rețea și sunt filtrate de cache/protocoale; pentru a aproxima mai bine traficul efectiv pe mediu, se recomandă exporturi NFS în mod sincron (sync), care forțează finalizarea pe mediu înaintea confirmării către client.

Al cincilea element este sistemul de monitorizare a parametrilor de funcționare a serverelor. Un modul dedicat urmărește resursele critice (CPU, memorie principală, încărcarea sistemului de operare) cu granularitate ridicată, utilizând utilitarul collectl [144]. Fișierele binare comprimate sunt ulterior convertite în rapoarte CSV pentru analiză. Această monitorizare este esențială pentru validarea rezultatelor, prin detectarea saturărilor de resurse care ar altera tiparele reale (de exemplu, saturația CPU pe un nod poate reduce artificial numărul de cereri I/O și volumul transferat).

Presupunând absența limitărilor CPU/RAM (validate de configurația fizică utilizată), rămân două componente care pot influența performanța și acuratețea măsurătorilor: (i) conectivitatea de rețea între nodurile de calcul și serverul de stocare, supusă constrângerilor fizice inerente; (ii) arhitectura logică a sistemului de stocare (partajare la nivel de bloc sau

fișier, tipul sistemului de fișiere), ale cărei opțiuni afectează latența, debitul și numărul de operații I/O. Pentru a elimina influențele mediilor persistente tradiționale, se propune utilizarea unui sistem de stocare bazat pe memoria volatilă a serverului de stocare, ca punct de referință „idealizat”.

3.3. Alegerea mediului de stocare și a platformei de virtualizare

În cadrul acestei secțiuni vor fi descrise etapele care au fost parcurse pentru a realiza alegerea unui sistem de stocare în memoria volatilă, luând în considerare alternativele care există în nucleul Linux. De asemenea, pentru a alegerea platformei de virtualizare care va fi folosită în cadrul fluxurilor experimentale este realizată o succintă analiză a modalităților pe care platformele open-source descrise în capitolul anterior le prezintă pentru monitorizarea evenimentelor din cadrul ciclului de viață al mașinilor virtuale.

3.3.1. Sisteme de stocare în memoria volatilă

3.3.1.1. Sistemul de fișiere *tmpfs*

Sistemul de fișiere *tmpfs* reprezintă un mecanism prin care se poate crea un spațiu de stocare în memoria volatilă (RAM), fiind utilizat de mult timp în cadrul sistemelor de tip UNIX [145]. Acest spațiu poate fi montat într-un director din ierarhia sistemului de fișiere, comportându-se ca un sistem de fișiere complet funcțional, fără a necesita suport de stocare persistent. Având în vedere că datele sunt stocate în RAM, acestea sunt, în mod evident, volatile și sunt pierdute în momentul opririi sau repornirii sistemului.

Funcționarea *tmpfs* se bazează pe alocarea dinamică a paginilor de memorie în spațiul de adresare al nucleului sistemului de operare, făcând spațiul disponibil proceselor utilizator. Un volum *tmpfs* se poate crea foarte ușor, printr-o singură comandă, iar dimensiunea maximă a unui volumului trebuie să fie specificată explicit la momentul montării [146]. O caracteristică importantă constă în faptul că, în condiții de solicitare intensă de resurse de memorie, nucleul Linux poate decide să mute anumite regiuni din *tmpfs* către spațiul de swap disponibil, asigurând astfel stabilitatea și continuitatea execuției proceselor. Acest tip de spațiu de stocare este utilizat frecvent în sistemele Linux pentru operațiuni temporare care necesită acces rapid, cum ar fi directoarele temporare utilizate exclusiv în timpul execuției sistemului, spațiile de lucru pentru compilatoare sau pentru stocarea intermediară în cadrul sarcinilor de procesare intensivă a datelor.

În contextul virtualizării și al containerizării, *tmpfs* este utilizat frecvent pentru izolarea datelor temporare sensibile sau pentru furnizarea unui spațiu de lucru cu performanță ridicată, care nu lasă urme persistente în sistem. În scenariul analizat în această lucrare, *tmpfs* este deosebit de util întrucât poate fi utilizat ca referință de performanță, sistemul fiind capabil să atingă limitele superioare ale subsistemului de memorie și ale canalelor de intrare/ieșire, așa cum au demonstrat și alți autori [147], [148].

3.3.1.2. *brd* - dispozitive de tip bloc în RAM

Nucleul Linux oferă dispozitivul virtual de tip bloc *brd* (Block RAM Disk), care pune la dispoziție un mediu de stocare volatil ce emulează comportamentul unui disc în accepțiune clasică, folosind exclusiv memoria RAM a sistemului. Spre deosebire de *tmpfs*, care expune direct un sistem de fișiere utilizabil, *brd* furnizează un dispozitiv brut de tip bloc, ce trebuie mai întâi formatat pentru a crea un sistem de fișiere, după care poate fi tratat identic cu un disc fizic. Discul virtual devine accesibil sub forma */dev/ramN*, unde *N* reprezintă indexul dispozitivului creat.

Din punct de vedere al implementării, modulul *brd* alocă un set de pagini de memorie contiguă în RAM și construiește imaginea internă a unui dispozitiv de tip bloc. Accesul la informațiile stocate se realizează prin operații clasice de bloc, citire, scriere, căutare, rutate prin straturile standard ale stivei de intrare ieșire, inclusiv *bio* și *blk-mk*, ceea ce permite testarea fidelă a interacțiunii dintre procese sau aplicații și infrastructura de stocare virtualizată astfel [45], [149]. Această arhitectură asigură integrarea transparentă cu utilitare uzuale pentru

discuri, precum mkfs, mount, dd, fio, și cu mecanisme de organizare de nivel superior, precum Logical Volume Manager, LVM, sau RAID software, md.

În literatura de specialitate, brd este utilizat frecvent ca dispozitiv de referință pentru experimentarea de strategii de planificare în subsistemul de intrare ieșire, pentru dezvoltarea de sisteme de fișiere experimentale și pentru validarea performanței subsistemului de bloc [45], [150]. Eliminând variabile externe, precum latențele de disc sau interferențele cu hardware-ul real, brd servește drept etalon pentru măsurători controlate.

În scenarii de tip HPC, High Performance Computing, sau în sisteme îmbarcate, embedded systems, brd este folosit pentru încărcarea rapidă a imaginilor de sistem sau a mediilor temporare cu rădăcina sistemului de fișiere în RAM, reducând timpii de pornire [151]. Totodată, în experimente dedicate caracterizării sistemelor de intrare ieșire în medii virtualizate sau containerizate, utilizarea unui disc virtual bazat pe brd elimină interferențele introduse de dispozitivele de stocare fizică, creând un mediu de măsurare determinist, adecvat testării.

3.3.1.3. ramfs și zram

Un sistem de fișiere minimalist utilizabil direct în RAM este ramfs, bazat exclusiv pe mecanismul de caching al paginilor din nucleul Linux [152]. Similar cu tmpfs, pentru orice fișier creat în ramfs se alocă memorie fără rezervă, ceea ce implică imposibilitatea eliberării spațiului prin mecanisme de swap; datele persistă în RAM până la eliberarea manuală a memoriei sau până la oprirea sistemului. Implementarea ramfs este foarte compactă, folosind infrastructura de caching a VFS și evitând copiile de pagini, ceea ce îi conferă eficiență în consumul de resurse, dar poate conduce la epuizarea memoriei dacă este utilizat necontrolat. Din acest motiv, doar utilizatorul root are permisiune de scriere, pentru a preveni erorile de tip OOM. Aplicațiile principale includ montarea unui sistem minimal de fișiere în RAM în faza de boot (initramfs/rootfs), pentru gestionarea rapidă a modulelor nucleului și a fișierelor temporare critice pentru pornire.

În 2014 a fost introdus în nucleul Linux modulul zram, care permite discuri virtuale în memorie cu compresie în timp real [153], concept propus anterior sub denumirea compcache [154]. Scopul principal este furnizarea unui spațiu de tip swap sau a unui disc virtual. Funcționarea se bazează pe comprimarea datelor la stocare și decompimarea la citire, cu execuția paralelă a fluxurilor de compresie și suport pentru algoritmi precum LZO-RLE, LZ4 și ZSTD [155]. Flexibilitatea configurării permite ajustarea raportului viteză/eficiență a compresiei în funcție de specificul sarcinilor. Utilizările uzuale includ crearea unui spațiu swap pentru recuperarea memoriei și stocarea temporară a fișierelor care necesită un acces mai rapid decât pe un mediu de stocare clasic.

3.3.2. Alegerea sistemului de stocare folosit pentru experimente

Pentru experimente s-a utilizat un spațiu de stocare virtual sub forma unui disc plasat în RAM, pentru a elimina limitările mediilor fizice și a obține măsurători cât mai fidele. Modulul zram a fost exclus deoarece comprimarea/decomprimarea în timp real adaugă întârzieri și consum de CPU, reducând performanța globală, chiar dacă diminuează volumul efectiv de date. Sistemul de fișiere ramfs a fost, de asemenea, exclus: bazarea pe mecanismele interne de caching ale nucleului ar introduce o variabilă greu controlabilă, capabilă să mascheze comportamente reale ale I/O. În consecință, comparația s-a limitat la tmpfs și brd, vizând lățimea de bandă și IOPS.

Testele au fost rulate cu fio (Flexible I/O Tester) [134], utilitar ce necesită corelarea atentă a rezultatelor cu parametrii de intrare pentru a evita interpretări eronate [156]. Indicatorii monitorizați au fost: lățimea de bandă (MB/s, GB/s) [157], IOPS (relevant pentru acces aleatoriu cu blocuri mici) [158], latența (ms, μ s) și adâncimea cozii (iodepth), a cărei ajustare optimizează performanța, mai ales pe SSD NVMe [157], [159]. Numărul mare de combinații posibile impune o agregare sistematică a datelor.

Memoria RAM, organizată pe bancuri, canale și ranguri, permite acces concurent la regiuni diferite și un paralelism masiv [160]. În acest context, iodepth a fost fixat la 16, fără a deveni un factor limitativ; literatura indică valori similare ca valide pentru sisteme rapide (de exemplu NVMe), fie ca minim funcțional (16), fie chiar 8 în alte studii [161], [33]. Printr-un script

automatizat s-au testat toate combinațiile pentru: (1) dimensiunea blocului (KB): 4, 8, 16, 32, 64, 128, 256, 512, 1024; (2) numărul de fire: 4, 8, 16, 32, 64, 128, 256, 512, 1024; (3) tipul accesului: randreadwrite. A fost reținut exclusiv accesul aleatoriu cu 50% citiri și 50% scrieri, adecvat mediilor virtualizate. Sunt prezentate în continuare doar rezultatele pentru lățime de bandă, IOPS și latențe, suficiente prin simetria observată la scriere. Graficele folosesc aceeași scară pe axa verticală pentru ambele soluții.

Se observă că brd furnizează rezultate superioare, în pofida necesității unui sistem de fișiere peste dispozitiv (în teste, ext4), cu lățimi de bandă aproape de limita memoriei RAM (DDR5 4800 MT/s). Diferențele provin din arhitectura internă: cu `--direct=1` (`O_DIRECT`), accesul ocolește cache-ul paginilor, transferând datele direct între spațiul utilizator și memoria fizică asociată dispozitivului brd, evitând dublarea copierilor și atingând performanțe foarte ridicate. Cu `--direct=0`, nucleul interceptează I/O și îl trece prin cache; pentru brd, aceasta duce la copieri inutile între memoria dispozitivului, cache și spațiul utilizator, implicând mecanisme suplimentare de blocare și degradând performanța. În cazul tmpfs, efectul `O_DIRECT` este neglijabil: tmpfs nu îl suportă nativ, iar încercările sunt ignorate sau convertite în acces prin cache, astfel că performanța rămâne în practică similară, la nivelul memoriei RAM.

Pentru evaluarea unui spațiu RAM utilizat ca sursă pentru un export NFS cu opțiunea `sync`, alegerea parametrilor `fi` este determinantă. Configurația adecvată este `--direct=0` combinată cu `--fsync=1`, care reproduce corect fluxul de date: cache normal urmat de sincronizări complete, similare efectului opțiunii `sync`. `--direct=1` este util doar pentru măsurarea performanței brute a dispozitivului RAM (brd), fără relevanță pentru un NFS funcțional. Rezultatele cu `--direct=0` și `--fsync=1` arată menținerea performanței pentru tmpfs și scăderea semnificativă pentru brd.

În NFS v4.1/v4.2, delegările, mecanismele de cache pe client și agregarea scrierilor influențează puternic latența și debitul observat de aplicații [140]. Rețeaua, RPC-urile și retransmisile pot deveni limitative, chiar dacă stocarea pe server este foarte rapidă. În acest cadru, un suport RAM lipsit de straturi suplimentare de planificare I/O și de cache paralele (tmpfs) oferă un comportament mai predictibil și ușor de interpretat: date stocate direct în memoria nucleului, integrare într-un singur nivel de cache, configurare simplă și export NFS imediat utilizabil. Prin urmare, pentru simularea unui spațiu RAM partajat prin NFS către mai mulți clienți se va utiliza tmpfs. Alegerea brd rămâne potrivită scenariilor specializate orientate spre analiză completă a stivei I/O, nealiniată scopului prezentei lucrări.

3.3.3. Instrumentarea monitorizării memoriei volatile partajate

Utilizarea tmpfs ca suport RAM pentru export NFS aduce o limitare metodologică majoră: nu există un dispozitiv de tip bloc asociat, astfel că nucleul nu produce statistici I/O standard în `/proc/sys`. Instrumente precum `collectl`, `dstat` sau `iostat` nu pot raporta volume/număr de operații, deoarece în tmpfs accesul are loc exclusiv în cache-ul de pagini, fără cereri bio. În Linux 6.8, acest comportament este intenționat: fișierele tmpfs sunt gestionate de subsistemul `shmem` (`mm/shmem.c`), iar operațiile trec prin `shmem_file_read_iter()` și `shmem_file_write_iter()` fără a implica cozi de I/O specifice dispozitivelor bloc.

Pentru a obține vizibilitate asupra traficului efectiv (citiri/scrieri) în acest context, a fost implementată o extensie de profilare pentru tmpfs, proiectată să respecte principiile nucleului (izolare, atomicitate, evitare dependențe circulare) și să se integreze nativ în infrastructura de expunere prin `sysfs`.

Arhitectura soluției: contoare la nivel de superblock (tmpfs): în structura `shmem_sb_info` au fost adăugate patru câmpuri `atomic64_t`: `read_bytes`, `write_bytes`, `read_operations`, `write_operations`. Inițializarea are loc în `shmem_fill_super`.

Expunere prin `sysfs`: crearea fișierului virtual `tmpfs_stats`, atașat la `kernel_kobj` în rutina `tmpfs_sysfs_init()` (apelată prin `module_init()`), pentru a publica contorizările la nivel de sistem de fișiere.

Deoarece `shmem` este, în practică, parte integrantă a nucleului, modificările necesită recompilarea kernelului și repornirea sistemului. Verificările preliminare au scos în evidență două neajunsuri care afectau acuratețea datelor:

- Citirile NFS peste RDMA necontorizate. Deși scrierile (cu export sync) erau contorizate corect, citirile efectuate de clienți prin NFS/RDMA ocoleau `shmem_*_iter` din cauza optimizărilor pe ruta splice din `nfsd`: datele din page cache erau mapate direct către adaptorul RDMA.
Remediu: introducerea unui punct suplimentar de contorizare în `nfsd4_encode_read` (`nfs4xdr.c`). După validarea succesului operației și identificarea tipului de FS, contoarele `shmem_sb_info` sunt actualizate. Această poziționare surprinde întreg volumul livrat clienților (inclusiv citiri parțiale/segmentate), indiferent de calea internă (copiere clasică vs. splice).
- Dimensiunea efectivă a blocurilor la nivel de VM. `tmpfs` raportează implicit acces pe pagini de 4 KB, nerelevând granularitatea reală a cererilor generate de VM-uri.
Remediu: extinderea modulului de colectare la nivel de hipervizor pentru a înregistra, per VM și la pas de 1 s, volumele transferate și numărul de operații pe direcții (read/write), calculând dimensiunea medie a blocului pe fereastra curentă. Valorile sunt diferențe incrementale (counter deltas) și se persistă într-un fișier separat, păstrând în paralel agregatele globale. Costul de execuție este neglijabil (operații în memorie, câteva actualizări de dicționare și o scriere/linie/VM).

Soluția furnizează o imagine completă și fidelă a traficului I/O pentru `tmpfs` exportat prin NFS:

- contoare robuste pentru citiri/scrieri la nivel de superblock, inclusiv citirile livrate prin RDMA;
- vizibilitate per-VM asupra volumelor, numărului de operații și dimensiunii medii a blocurilor, sincronizată temporal cu restul metricilor din infrastructură.

Prin această abordare, se depășește opacitatea structurală a `tmpfs`, păstrând totodată compatibilitatea cu fluxurile anterioare de analiză și un overhead practic inexistent, ceea ce permite profilarea precisă a tiparelor de acces într-un mediu virtualizat care folosește RAM ca suport de stocare.

3.3.4. Alegerea platformei pentru managementul cloud-ului

Toate cele trei platforme analizate, OpenNebula, OpenStack și CloudStack, oferă capabilități similare pentru orchestrarea resurselor virtualizate pe hardware standard, fără cerințe dedicate. Diferențele apar însă la nivelul arhitecturii interne, al gradului de modularitate și al modului de integrare cu sisteme externe. OpenNebula se remarcă printr-o arhitectură centralizată, coerentă și simplă, în care un serviciu principal orchestrează întregul ciclu de viață al resurselor, sprijinit de o interfață web matură și de un CLI bine integrat; această simplitate o face deosebit de potrivită pentru laboratoare de cercetare și universități, unde timpul de implementare și ușurința operării sunt critice [168]. OpenStack, în schimb, este o platformă distribuită și profund modulară, cu servicii autonome interconectate prin mesagerie asincronă; oferă flexibilitate și scalabilitate la scară foarte mare, dar la prețul unei complexități crescute în instalare, integrare și mentenanță [169]. CloudStack se poziționează între cele două: model centralizat de orchestrare, management server cu agenți pe hosturi și o bază de date unificată; instalarea și operarea sunt relativ simple, însă extensibilitatea este mai limitată și orientată cu precădere spre plugin-uri Java. În practică, alegerea depinde nu doar de criterii tehnice, ci și de contextul instituțional, disponibilitatea expertizei și nivelul de control dorit asupra infrastructurii [170].

Pentru obiectivul acestei lucrări, monitorizarea evenimentelor din ciclul de viață al mașinilor virtuale, este esențială existența unor mecanisme de tip „hook” (acțiuni automate declanșate la apariția unor stări cheie). OpenNebula oferă nativ hook-uri la nivelul serviciului central (oned), cu suport pentru multiple obiecte (VM, host, imagini, utilizatori, VDC etc.) și două clase de declanșatoare: pe stare (STATE) și periodice (STATE_CHANGE + TIMEOUT). Execuția este integrată în fluxul intern al orchestratorului, ceea ce asigură fiabilitate și sincronizare. OpenStack nu are un mecanism de hook-uri unificat, dar permite reacții automate prin: (i) sistemul de notificări al serviciilor, ce poate fi interceptat pentru acțiuni externe și se integrează cu subsistemul de telemetrie [171]; (ii) fluxuri de lucru definite în Mistral, pentru acțiuni condiționale [172]; (iii) alarme Aodh și orchestrare Heat, care pot declanșa acțiuni la praguri/condiții. Aceste opțiuni acoperă o parte din nevoile de automatizare, însă nu sunt

sincronizate nativ cu toate stările interne ale instanțelor. CloudStack nu furnizează hook-uri utilizator direct, dar oferă extensibilitate prin plugin-uri Java în Management Server [173], un „Event Bus” pentru observarea evenimentelor [174] și unele webhook-uri/callback-uri limitate [175]. În ansamblu, pentru control fin și reacții la stări VM, OpenNebula dispune de cea mai directă și robustă abordare.

Un alt criteriu important, dată fiind utilizarea unui backend de stocare în RAM cu capacitate limitată, îl reprezintă eficiența în administrarea spațiului de stocare. Tehnologia Copy-on-Write (CoW) optimizează consumul de spațiu prin partajarea blocurilor și copierea lor doar când apar modificări [42]; literatura arată beneficii clare în timpii de clonare, snapshot și backup, dar și costuri de management al metadatelor și potențiale penalizări în scenarii cu scrieri intense și fragmentate [40]. În acest context, formatul QCOW2 (QEMU Copy-on-Write v2) este deosebit de relevant: permite instanțierea VM-urilor dintr-o imagine de bază fără copiere imediată a datelor, crescând progresiv doar pe măsura scrierilor [176]. Există însă un overhead inerent operațiilor CoW, care trebuie ponderat în funcție de profilul de lucru.

OpenNebula integrează nativ și simplu imaginile QCOW2 și exploatează eficient CoW pentru instanțiere rapidă din imagini de referință, fără clonerizare completă; lanțurile de imagini rămân scurte, păstrând un echilibru bun între performanță și economie de spațiu [177].

Având în vedere cerințele metodologice ale experimentelor (hook-uri fiabile pe stări VM, timp de implementare redus, eficiență în utilizarea spațiului de stocare în RAM) și atuările operaționale descrise, platforma aleasă pentru implementarea arhitecturii de test este OpenNebula. Aceasta oferă combinația optimă între simplitatea administrării, extensibilitate orientată pe evenimente și eficiență în gestionarea imaginilor QCOW2, facilitând lansarea rapidă a unui număr mare de instanțe pe spațiu limitat, cu un efort operațional minim.

3.3.5. Configurarea și optimizarea sistemelor

Pentru desfășurarea experimentelor tehnice în condiții cât mai bune și controlate, au fost aplicate o serie de optimizări preliminare asupra sistemelor de operare, având ca obiectiv îmbunătățirea performanței generale în ceea ce privește operațiile de comunicație în rețea și execuția proceselor. Deși nu s-a intervenit asupra mecanismelor interne de foarte jos nivel, ajustările implementate au fost orientate strategic spre asigurarea transferurilor de date cu latență cât mai mică și lățime de bandă cât mai mare dar și pentru sporirea eficienței execuției instanțelor virtuale. Aceste optimizări au fost aplicate uniform la nivelul tuturor nodurilor de găzduire și a serverului care asigură suportul de stocare prin discul virtual în memoria RAM.

3.3.5.1. Optimizarea frecvenței de lucru a procesoarelor

Distribuțiile Linux ajustează dinamic frecvența procesorului pentru echilibru între consum și performanță. În experimente frecvența a fost fixată la nivelul maxim cu cpupower și guvernatorii ondemand conservative și schedulul au fost dezactivați [180]. Obiectivul este un mediu predictibil și reproductibil fără variații de frecvență în timpul măsurărilor. Guvernatorul performance menține frecvența la valoarea maximă pe toată durata execuției conform documentației Linux [181]. Variațiile de frecvență pot afecta rezultatele în teste de intrare ieșire și în scenarii sensibile la latență. Stabilizarea frecvenței reduce variabilitatea între rulări și sprijină rezultate reproductibile. Fără această setare sistemul schimbă frecvența în funcție de factori greu de observat și poate influența măsurătorile. Pentru experimente se recomandă folosirea guvernatorului performance în special la aplicații scurte sau sensibile la latență [182].

3.3.5.2. Optimizarea parametrilor de rețea

Optimizarea vizează latență mică și lățime de bandă mare pentru HPC centre de date și virtualizare [183]. Transportul este RoCE v2. QoS se configurează unitar pe NIC și pe switch. Pe nodurile cu Mellanox ConnectX 4 la 25 Gbps se folosește mlx_qos din pachetul de drivere Nvidia [184]. Parametrii se importă automat din switch prin DCBX în modul firmware cu LLDP. Traficul de stocare este izolat într-un VLAN dedicat. Toate prioritățile 802.1p sunt mapate la clasa 3 la intrare și la ieșire pe VLAN 331. Se folosește o singură clasă PFC pentru simplitate și consistență. Switch-ul aplică DCB IEEE cu un profil care alocă 86% din bandă pentru clasa 3. Restul rămâne pentru best effort și rezerve mici. Pe serverul de stocare cu Intel E810 C

DCBX nu a aplicat setările în firmware. Configurația PFC și ETS s-a introdus manual cu Ildptool și este aliniată cu switch-ul. RoCE v2 funcționează simetric pe toate porturile.

3.3.5.3. Configurări și optimizări în OpenNebula

Planificatorul OpenNebula plasează și pornește VM-urile dintr-o coadă FIFO, controlat de SCHED_INTERVAL, MAX_VM, MAX_DISPATCH și MAX_HOST. MAX_VM limitează numărul de cereri analizate pe ciclu, MAX_DISPATCH stabilește câte VM-uri se lansează efectiv, iar MAX_HOST restrânge numărul de instanțe per nod într-o rundă, pentru a evita aglomerarea. Fluxul pe ciclu este: selectarea cererilor (MAX_VM), asocierea cu noduri eligibile și aplicarea constrângerilor locale (MAX_HOST), apoi lansarea în limita MAX_DISPATCH, ceea ce controlează încărcarea și distribuția.

Eficiența operațiilor cu imagini și discuri depinde și de TM_MAD și DATASTORE_MAD; parametrul „-t” fixează numărul de fire paralele atât pentru transferuri (one_tm), cât și pentru operații locale pe imagini (one_datastore). Valori mari accelerează execuția, dar trebuie calibrate hardware-ului; o valoare precum 15 oferă un compromis între paralelism și stabilitate. Pentru testele experimentale s-au urmărit distribuția uniformă a VM-urilor. S-a setat SCHED_INTERVAL=1, MAX_VM=5000, MAX_DISPATCH=0 și, critic, MAX_HOST=1, astfel încât într-un ciclu fiecare nod primește cel mult o instanță, obținându-se echilibrare progresivă. Orchestrarea pornește loturi pentru două modele (Linux/Windows) cu oneinstantiated, lansarea reală respectând scheduler-ul.

Un al doilea scenariu, de stres, a maximizat paralelismul: „-t” a fost crescut la 80, MAX_VM=5000, MAX_DISPATCH=0, iar MAX_HOST=50, pentru încărcări concentrate pe noduri. Deoarece algoritmul standard tinde să dezechilibreze plasarea, scriptul de instanțiere a fost modificat pentru a selecta explicit nodul la inițializare și s-a trecut la un script Python care comunică prin RPC cu orchestratorul, reducând drastic latența programării. Integrarea cu sistemul extern de monitorizare s-a realizat prin două hook-uri native, pe PROLOG („deploy”) și BOOT („boot”), care raportează momentele cheie și nodul de găzduire.

3.4. Concluzii intermediare

Capitolul motivează și descrie o arhitectură experimentală pentru identificarea tiparelor de I/O în platforme de virtualizare, plecând de la necesitatea analizării accesului la disc dincolo de volumul total (distribuție temporală, dimensiuni de bloc, număr de operații), cu impact direct asupra debitului și latențelor. Exemplele din literatură arată variația raportului scrieri/citiri în funcție de server și software, influența dimensiunii cererilor și a paralelismului, precum și diferențele dintre stocarea locală NVMe și sisteme de bloc în rețea, concluzionând că tiparele trebuie măsurate în mediul propriu înainte de producție. Arhitectura logică propusă monitorizează simultan hipervizoarele (agregat pe VM) și serverul de stocare (perspectiva sistemului), corelează temporal evenimentele și metricile și exportă rezultatele în CSV, cu un flux de lucru care acoperă inițializarea, colectarea, sincronizarea și generarea de grafice.

Ciclul de viață al VM-urilor este formalizat prin stările „instantiated”, „deploy”, „boot”, „ready”, „start”, „end”, raportate către un server web (/status, /go-status) cu semafor pentru coordonare; orchestratorul lansează, supraveghează, reface instanțe eșuate, oprește și curăță resursele și raportează timpii. Pe hipervizoare, colectarea la o secundă urmărește operații, volume, latențe și dimensiunile blocurilor; la server, monitorizarea este realizată prin adaptări specifice nucleului Linux. Arhitectura hardware include un server de stocare, opt noduri de calcul, servere pentru orchestrare și fișiere și rețele segregate pe un switch L2 cu interfețe de mare viteză. Pentru a elimina constrângerile mediilor persistente se folosește stocare în RAM (tmpfs, brd), cu detalii de configurare și utilizare, iar monitorizarea în nucleu este instrumentată prin ajustări ale subsistemelor de memorie partajată și NFS. Alegerea platformei de virtualizare este justificată în raport cu cerințele experimentului. Capitolul pregătește astfel metodologia de lucru pentru testele practice și definirea fluxurilor experimentale de măsurare, colectare și analiză.

Capitolul IV - Metodologie pentru realizarea testelor experimentale

Studiul urmărește definirea unui profil detaliat de performanță pentru subsistemul de stocare din infrastructuri de virtualizare, folosind un mediu bazat pe memorie volatilă pentru a evita congestiile asociate stocării tradiționale. Profilul vizează IOPS, lățimea de bandă, latențele de citire/scriere, raportul citiri-scrieri și repartitia dimensiunilor blocurilor, indicatori evaluați atât la nivelul hipervizoarelor, cât și la nivelul stocării în RAM, pentru a evidenția eventualele discrepanțe introduse de caching, optimizări software sau specificul protocoalelor (de tip NFS cu RDMA). Un obiectiv central este caracterizarea proporției dintre scrierile și citirile reale la suportul de stocare și solicitările generate de mașinile virtuale, precum și a repartitiei operațiilor în funcție de dimensiunea blocurilor, în condițiile în care mecanismele de agregare și writeback pot masca fluxurile reale. În acest scop, se propune o metodologie de măsurare directă la nivelul stocării în RAM, prin instrumentații integrate în nucleul Linux, pentru cuantificarea precisă a operațiilor la sursă. Complementar, studiul își propune să determine valorile medii și maxime necesare pentru lățimea de bandă și IOPS ale sistemului de stocare într-un mediu educațional, astfel încât mașinile virtuale să funcționeze în parametri optimi.

4.1. Justificarea necesității testării experimentale

Lucrarea caută configurația optimă pentru un sistem de stocare partajat destinat unei platforme de virtualizare printr-o metodologie experimentală bazată pe un profil de intrare ieșire obținut în prealabil. Profilul se determină pe un disc virtual în memorie în scenarii reprezentative. Se măsoară dimensiunile blocurilor proporția dintre citiri și scrieri și nivelul de paralelism. Acest tipar devine referință pentru toate testele ulterioare.

Urmează testarea pe un sistem de stocare real din mediul de virtualizare. Se compară niveluri RAID software sisteme de fișiere locale partajate și soluții integrate precum ZFS sau infrastructuri iSCSI cu sisteme de fișiere pentru acces concurrent. Scopul este analiza comportamentului față de profilul obținut nu obținerea unor vârfuri teoretice. Sunt discutate diferențe între RAID 0 RAID 5 RAID 6 și RAID 10. RAID 0 oferă lățime de bandă mare fără redundanță. RAID 5 și RAID 6 adaugă toleranță la defecte cu penalizări la scriere. RAID 10 oferă un echilibru între performanță și reziliență cu spațiu utilizabil redus. Validarea se face prin teste legate de profil.

Evaluarea include și sisteme de fișiere. Ext4 XFS și ZFS folosesc politici distincte de alocare jurnalizare și caching. ZFS aduce verificare a integrității și replicare cu costuri de procesare mai mari. Fără teste adaptate profilului selecția ar fi arbitrară. Sunt luate în calcul și soluții distribuite. ZFS ca sistem complet sau iSCSI cu OCFS2. Performanța depinde de latența și lățimea de bandă a rețelei și de gestionarea accesului concurrent. Aceste efecte se evaluează prin teste directe.

Obiectivul este alegerea combinațiilor care se potrivesc profilului și folosesc eficient resursele. Se urmărește stabilitatea în timp. Platformele educaționale au nevoie de comportament previzibil sub încărcare. Testele pot arăta fluctuații și pot selecta variante stabile. Interacțiunile cu hipervizoarele și cu formatele de discuri virtuale sunt analizate. QCOW2 aduce alocare dinamică și metadata care pot afecta latența. Doar măsurarea directă arată dacă optimizările interne compensează aceste efecte.

Metodologia include teste de scalabilitate. Se verifică evoluția performanței când crește numărul de instanțe. Se integrează și analiza consumului de resurse pentru a evita configurațiile care solicită excesiv procesorul memoria sau rețeaua. Profilul de intrare ieșire funcționează ca filtru pentru selecția scenariilor. Testele devin instrument de diferențiere între opțiuni. Procesul este iterativ. Se colectează profilul pe stocare simulată. Se testează combinații pe platforma reală. Se corelează performanța cu resursele consumate. Se evaluează consistența și scalabilitatea. Rezultatul este o configurație adecvată mediului și sarcinii de lucru. Abordarea oferă o bază empirică pentru proiectarea infrastructurii. Testarea controlată și profilul derivat din scenarii reale permit alegerea unei arhitecturi de stocare cu funcționare previzibilă și utilizare echilibrată a resurselor.

4.2. Structura testelor și modalități de testare

4.2.1. Configurația mașinilor virtuale

Sunt folosite două tipuri de mașini virtuale. Mașinile virtuale Linux folosesc Ubuntu Server 24.04 LTS. Au 4 vCPU. Au 2 GB RAM. Au disc virtual de 8 GB. Generează acces aleatoriu cu operații scurte și mici. Mașinile virtuale Windows folosesc Windows 11 24H2. Au 8 vCPU. Au 8 GB RAM. Au disc virtual de 40 GB. Necesită interfață grafică accesibilă de la distanță. Produc un tipar mixt cu acces secvențial și aleatoriu. Configurațiile permit evaluarea performanței stocării și a influenței sistemului de operare asupra lățimii de bandă asupra latenței și asupra IOPS.

Mașinile virtuale sunt distribuite uniform pe nodurile de găzduire. Se evită supraîncărcarea unor noduri. Se obțin condiții comparabile între scenarii. Lansarea și oprirea sunt automatizate printr-un script de management și control. Monitorizarea în timp real este inclusă. Automatizarea reduce erorile umane. Rezultatele devin repetabile. Configurația reflectă o infrastructură de producție. Există mai multe noduri și sisteme de operare virtualizate. Instrumentarea din nucleul Linux măsoară traficul efectiv în memoria volatilă. Se obține o perspectivă directă asupra operațiilor fizice de intrare ieșire.

4.2.2. Moduri de testare

Un test pornește cu lansarea unui număr prestabilit de instanțe distribuite uniform pe toate nodurile. Scopul este o încărcare simetrică a resurselor. După boot fiecare instanță rulează un script adaptat sistemului de operare.

În Linux scriptul face actualizări complete. Instalează un nucleu nou. Repornește instanța. Rulează un nou set de actualizări. Instalează două grupuri de pachete. Include utilitare. Include două servere de baze de date. Include un server web. Descarcă un fișier cu aproximativ 200000 de comenzi SQL. Populează o bază de date MySQL în aceeași instanță. Etapele sunt urmărite prin fișiere cu rol de semafor din cauza repornirilor dintre pași. În Windows scriptul descarcă și instalează două IDE-uri. Apoi instalează pachetul WAMP cu toate bibliotecile necesare. Totul se desfășoară automat fără interacțiune în interfața grafică.

Pornirea scripturilor este automată prin mecanismele native. În Linux rulează prin cron. În Windows rulează prin Task Scheduler. Scripturile trimit asincron starea către serverul de monitorizare. Stările raportate sunt ready, start și end. După starea ready scriptul intră într-o așteptare controlată de o variabilă binară. Variabila este citită de la serverul web al monitorizării prin REST. Când valoarea devine 1 execuția continuă. Astfel se pot sincroniza etapele în multe instanțe.

Există două moduri de testare. Modul mediu dezactivează semaforul. După ready și între pași apar întârzieri aleatorii între 5 și 60 de secunde. Rezultatul este o încărcare moderată și asincronă. Modul maxim folosește semaforul. Execuția pornește aproape simultan în toate instanțele într-un interval de aproximativ 1 secundă. Se obține o încărcare concentrată asupra stocării. La final scripturile trimit comanda de oprire. Se evită procese automate nedorite. Se închid ordonat toate operațiile de intrare ieșire. Datele ajung pe suportul persistent. Se asigură un comportament previzibil pentru măsurare.

4.2.3. Organizarea seriilor de teste

În virtualizare discurile pot fi în formate diferite. Acestea influențează consumul de resurse și performanța. Formatul raw este o copie bit cu bit a unui disc fizic. Dimensiunea fișierului este egală cu dimensiunea logică a discului. Spațiul este ocupat chiar dacă datele reale sunt puține. La instanțiere fișierul este duplicat integral pentru fiecare instanță. Rezultatul este timp mai mare de lansare și ocupare ridicată a stocării.

OpenNebula folosește și formatul qcow2 [178]. Acesta aplică Copy on Write. La instanțiere se păstrează o referință la imaginea de bază. Se salvează doar blocurile modificate. Spațiul ocupat crește în funcție de datele scrise. Avantajul este clar când stocarea este limitată în RAM. Se pot lansa mai multe instanțe în paralel decât cu raw. Metodologia include două scenarii. Unul cu imagini raw. Unul cu imagini qcow2. Se testează patru serii. Încărcare medie

cu qcow2. Încărcare medie cu raw. Încărcare maximă cu qcow2. Încărcare maximă cu raw. Pentru fiecare serie există trei seturi. Set Linux cu 16 până la 320 instanțe. Set Windows cu 16 până la 160. Set mixt cu 16 până la 240. Aceste limite sunt pentru qcow2. Pentru raw limitele maxime sunt mai mici. Limita superioară depinde de memoria RAM folosită ca stocare și de dimensiunea discurilor virtuale ale instanțelor.

Numărul de instanțe este ales în multipli de 16. Distribuția este astfel echilibrată pe cele 8 noduri. Alegerea urmează și practici din centre de date. Această stratificare permite analiza diferențelor între sisteme de operare și a efectului sarcinilor mixte asupra tiparului de acces la stocare.

4.2.4. Fluxul experimental

Fiecare experiment urmează pași clari. Se pornesc sistemele de monitorizare pe hipervizoare și pe serverul de stocare. Astfel se captează întreg ciclul de viață al fiecărei instanțe. Se rulează scriptul de management și control. Acesta pornește serverul web pentru stări. Lansează mașinile virtuale după configurația curentă. Distribuie uniform instanțele între noduri când plasarea este manuală. Urmărește progresul fiecărei instanțe. Șterge instanțele la final. Se opresc sistemele de monitorizare la finalul rulării. Urmează centralizarea și agregarea datelor. Se colectează loguri. Se colectează fișiere CSV de pe hipervizoare. Se colectează metadate despre configurația rulării. Datele sunt apoi procesate. Se generează reprezentări grafice. Se calculează valori medii și maxime pentru metricile urmărite. Pașii unu, doi și trei sunt automatizați printr-un script. Fiecare set de teste se repetă de trei ori. Se acceptă doar rezultate cu variație sub zece la sută.

Metodologia crește treptat numărul de instanțe. Astfel se pot identifica praguri de scalabilitate. Se poate lega încărcarea de performanță. Se obțin medii mai robuste pentru lățime de bandă și pentru IOPS. Rezultatele reflectă scenarii reale din medii cloud educaționale.

4.3. Mărimi măsurate și metrici de performanță

Pentru a construi o imagine completă și precisă asupra performanței subsistemului de stocare bazat pe memoria RAM și a modului în care acesta interacționează cu infrastructura virtualizată, metodologia experimentală adoptată în acest studiu prevede măsurarea unui set complex de metrici, selectate astfel încât să surprindă toate dimensiunile relevante ale comportamentului sistemului.

4.3.1. Măsurători temporale la nivelul mașinilor virtuale

Element central al colectării îl reprezintă monitorizarea momentelor-cheie din ciclul de viață al fiecărei VM: „instantiated”, „deploy”, „boot”, „ready”, „start” și „end”. Înregistrarea lor precisă are o dublă utilitate: (1) permite evaluarea platformei de virtualizare prin cuantificarea timpilor pentru fiecare etapă și (2) asigură corelarea metricilor de performanță între nivelul hipervizorului, subsistemul de stocare și instanțele virtuale, facilitând identificarea legăturilor cauzale. Reprezentarea grafică a acestor marcaje evidențiază rapid eventualele funcționări neadecvate ale unor VM-uri; în astfel de cazuri, testul se exclude și se reia.

4.3.2. Metrici de performanță la nivelul hipervizoarelor

La nivelul hipervizoarelor se colectează metrici care surprind perspectiva sistemului de virtualizare asupra stocării, agregând cererile tuturor VM-urilor. Sunt măsurate lățimile de bandă la citire și scriere (MB/s), IOPS la citire și scriere, precum și latențele medii pentru fiecare tip de operație, utile pentru evaluarea calității serviciului perceput de aplicații. În plus, se înregistrează dimensiunile blocurilor de citire/scriere pentru fiecare VM, care sunt ulterior agregate într-o repartitie pe categorii standard (512B, 1KB, 2KB, 4KB etc.) pentru a caracteriza tiparul efectiv de acces.

4.3.3. Măsurători la nivelul subsistemului de stocare

Studiul introduce o contribuție metodologică originală: instrumentarea nucleului Linux pentru a monitoriza direct operațiile fizice de citire/scriere asupra memoriei virtuale, validând gradul de corelare dintre fluxurile de I/O raportate de hipervizoare și traficul efectiv procesat de stocare. La nivelul discului virtual din RAM, partajat prin NFS+RDMA, se măsoară lățimea de bandă la citire/scriere (globală, cât mai puțin influențată de caching) și IOPS la citire/scriere, înregistrate direct în subsistemul de stocare (și serviciul NFS). Compararea nivelului virtual cu cel fizic permite identificarea discrepanțelor și a optimizărilor implicite (ex. caching agresiv, delayed allocation sau operații derulate strict la nivelul stocării).

4.3.4. Monitorizarea resurselor hardware

Metodologia completează măsurătorile de stocare cu monitorizarea continuă a resurselor hardware pentru a corela intensitatea I/O cu gradul de încărcare al infrastructurii și a identifica congestiile care pot distorsiona rezultatele. Pe nodurile de găzduire și pe serverul de stocare se urmăresc: %CPU, load average (indicând supra-aglomerarea cozilor) și memoria utilizată, excluzând spațiul ocupat de tmpfs pentru a izola consumul real. Traficul de rețea este exclus în mod deliberat din analiză, deoarece RDMA nu oferă raportări consecvente prin utilitare standard, ceea ce ar introduce ambiguități în interpretare.

4.3.5. Sincronizarea temporală

Metodologia impune sincronizarea strictă a timpului pe toate nodurile, folosind NTP pentru alinierea ceasurilor serverului de stocare și a hipervizoarelor la nivel de milisecundă (sau microsecundă, când hardware-ul permite). Aceasta face posibilă corelarea temporală a datelor colectate în puncte diferite ale arhitecturii, asigură coerența reprezentărilor grafice și permite asocierea unor evenimente (de exemplu vârfuri de latență) cu acțiuni precise din VM-uri. În plus, sincronizarea susține validarea ipotezelor de cauzalitate în scenarii de congestie. Măsurarea simultană și corelată a metricilor oferă o înțelegere solidă a comportamentului stocării în RAM sub sarcini reale și fundamentează concluziile experimentale.

4.3.6. Diagrame la nivel de set de teste (agregare pe rulări)

Rezumatul setului de teste urmărește două ținte. Mai întâi se verifică omogenitatea distribuirii încărcării și regularitatea ciclurilor de viață. Apoi se cuantifică metricile în ferestrele dintre starea start și starea end. Exemplele provin dintr-un set cu 320 de mașini virtuale Linux cu disc QCOW2 și mod de testare maximă. Harta de căldură arată repartizarea spațială a instanțelor pe noduri și pe rulări. Un model uniform indică o plasare echilibrată. Benzi mai întinse pe anumite noduri pot semnala concentrarea instanțelor. Diagrama duratelor agregă mediile pentru boot start și end și adaugă benzi cu valori minime și maxime. Lățimi mari ale benzilor pot indica dispersie ridicată sau comportamente atipice. Aceeași reprezentare ajută la trierea rulărilor cu abateri mari.

Diagramele pentru lățimea de bandă și pentru IOPS afișează medii pe direcții și vârfuri instantanee. Curbele sunt agregate pe rulări și pe numărul de instanțe. Separarea pe puncte de monitorizare permite compararea dintre ceea ce vede stocarea și ceea ce emit instanțele. Vârfurile arată amplitudinea maximă atinsă în timpul sarcinilor.

Raportul dintre scriere și citire este urmărit atât pentru lățime de bandă cât și pentru IOPS. Curbele indică dominanța uneia dintre direcții și oferă ponderea folosită la definirea profilului. Hărțile termice temporale prezintă dimensiunile blocurilor pe axa verticală și timpul pe axa orizontală. Intensitatea culorilor reflectă frecvența relativă a fiecărei dimensiuni la nivelul hipervizoarelor.

Distribuția dimensiunilor blocurilor este redată prin bare grupate. Procentele sunt normalizate pe citire și pe scriere și agregate pe toate nodurile. Reprezentarea face vizibilă forma repartiției pentru întregul set de teste.

4.3.7. Diagrame la nivel de rulare (diagnostic și încadrare)

La nivelul fiecărei rulări se generează diagrame care verifică evoluția corectă și delimitează exact intervalul de execuție al sarcinii. Aceste diagrame decid ce date intră în agregări și explică eventuale deviații locale.

Prima diagramă arată cronologia ciclurilor de viață ale fiecărei mașini virtuale. Fiecare instanță apare pe o linie cu segmente colorate pentru stări. Alinierea segmentelor start indică sincronizarea. Alinierea slabă arată întârzieri. Stări boot prea lungi sau end prea devreme duc la excluderea rulării. Scara timpului poate avea două rezoluții pentru a evidenția începutul.

A doua diagramă arată seriile temporale ale lățimii de bandă la citire și la scriere în ambele puncte de monitorizare. Fereastra de lucru este marcată clar. Mediile pe interval sunt trasate ca linii orizontale pentru toate cele patru mărimi. O a treia diagramă repetă aceeași logică pentru IOPS. Marcajele start și end și mediile sunt păstrate.

O altă diagramă prezintă încărcarea procesoarelor pe nodurile de virtualizare. Aceasta face vizibile momentele de congestie la nivel de nod. Urmează diagrama pentru gradul de încărcare a memoriei pe aceleași noduri. Aceasta indică situațiile în care memoria devine insuficientă și poate întârzia execuția.

Ultima diagramă arată utilizarea totală a procesoarelor pe serverul NFS. Toate nucleele sunt incluse. Graficul ajută la detectarea momentelor cu congestie care pot afecta măsurătorile.

4.4. Metodologie pentru determinarea profilurilor de intrare/ieșire

Testele sunt grupate pe scenarii. Fiecare scenariu are două seturi. Fiecare set are trei rulări identice. Pentru fiecare rulare se generează toate diagramele necesare. Dacă apar abateri sau întreruperi rulare se reface. După validare toate măsurile se mediează pe cele trei rulări. Rezultatele se exportă în fișiere CSV standard.

Distribuția dimensiunilor blocurilor se obține prin discretizare pe puteri ale lui 2. Intervalul este de la 512 B la 2 MB. Alocarea este exclusivă și urmează pragurile în ordine. Rezultatul dă procente stabile pe categorie pentru citire și pentru scriere. Aceste valori pot fi folosite direct în fio. CSV-urile pentru distribuții includ numărul de pachete pe categorie și procentul aferent. Fișierele sumar includ mediile indicatorilor măsurați.

Fluxul asigură controlul calității pe rulare. Asigură medierea pe trei execuții. Asigură formate CSV unitare. Asigură profilarea pe dimensiuni de bloc pe ambele direcții. Datele sunt apoi pregătite pentru pașii următori ai metodologiei.

Sunt definite două profiluri de intrare ieșire. Primul este la nivelul hipervizoarelor și reflectă comportamentul unui nod cu instanțe active. Al doilea este la nivelul serverului NFS și condensează traficul agregat din toate nodurile. Profilul de hipervizor este util când se simulează un singur nod. Profilul NFS pune sarcina totală direct pe sistemul de stocare și simplifică logistica. Profilul NFS nu oferă detalii pe nod.

4.4.1. Cadrul general al mărimilor măsurate și convenții de notare

Profilul de intrare ieșire este definit prin ponderea operațiilor de citire din total și prin ponderea operațiilor de scriere ca diferență până la unu. Include ponderea lățimii de bandă la citire din total și ponderea la scriere ca diferență până la unu. Include repartitia pe clase de dimensiuni de bloc pentru citire și pentru scriere. Notarea P_{IO} folosește α_{read} și β_{read} , care se obțin ca media IOPS la citire împărțită la suma mediilor IOPS la citire și la scriere, iar α_{write} este unu minus α_{read} . β_{read} se obține ca media lățimii de bandă la citire împărțită la suma mediilor la citire și la scriere. Aceste date permit configurarea exactă în fio [134].

Pentru un set de teste se definesc valorile medii pe o fereastră de lucru de la t_{start} la t_{end} . La serverul NFS se folosesc mediile pentru lățimea de bandă la citire și la scriere în MB pe secundă. Se folosesc mediile pentru IOPS la citire și la scriere. La nivelul hipervizoarelor se folosesc aceleași tipuri de medii. Se colectează și numărul de blocuri pe clase de dimensiuni la citire și la scriere. Clasele sunt ordonate și exprimate în octeți. Repartițiile medii pe clase se normalizează astfel încât suma pe fiecare direcție să fie unu. Lista categoriilor merge de la

512 B la 2 MiB în pași pe puteri ale lui doi. Valorile reprezentative sunt 2 la puterea 9 până la 2 la puterea 21 în octeți.

Din măsurători se obțin direct mediile la hipervizoare pentru IOPS și lățime de bandă și numerele de blocuri pe clase. Se poate construi profilul P_{10} la nivel de hipervizor. La serverul NFS există doar mediile pentru lățime de bandă și pentru IOPS. Nu există repartiții pe clase. Pentru a obține un profil complet la NFS se propun două metode matematice. Acestea pornesc de la corelația cu repartițiile de la hipervizoare și respectă ponderile măsurate la NFS pe fiecare direcție. Rezultatul dă repartițiile pe clase la NFS și permite definirea profilului P_{10} la NFS. Profilul astfel obținut se poate folosi în fio pentru teste direct pe serverul de stocare fără clienți.

4.4.2. Aproximarea liniară a repartițiilor blocurilor la nivelul sistemului de stocare

Metoda descrie cum aproximăm repartiția dimensiunilor de bloc la nivelul NFS folosind doar informațiile măsurate. Se pornește de la lista de dimensiuni posibile ale blocurilor. Pentru fiecare dimensiune există o pondere măsurată la hipervizoare. La NFS sunt cunoscute doar media debitului și media numărului de operații pe secundă. Din aceste mărimi se obține media dimensiunii blocului la NFS. Scopul este să găsim o repartiție pe aceleași dimensiuni care are suma ponderilor egală cu unu și are media egală cu media de la NFS.

La hipervizoare se calculează și media lor. Din repartiția măsurată se construiesc două repartiții auxiliare. Prima pune accent pe blocurile mari. A doua pune accent pe blocurile mici. Prima are o media mai mare decât media de la hipervizoare. A doua are o media mai mică decât media de la hipervizoare.

Se combină liniar repartiția măsurată cu una dintre repartițiile auxiliare. Se folosește un factor cu valori între zero și unu. Dacă trebuie crescută media se alege repartiția orientată spre blocuri mari. Dacă trebuie scăzută media se alege repartiția orientată spre blocuri mici. Factorul se alege astfel încât media noii repartiții să fie egală cu media de la NFS. În acest fel se obține o repartiție validă la NFS pe aceleași clase de dimensiuni.

Metoda funcționează dacă media de la NFS se află între media repartiției orientate spre blocuri mici și media repartiției orientate spre blocuri mari. Dacă media de la NFS iese din acest interval se poate modifica treptat repartiția măsurată la hipervizoare. Se mută o parte din pondere de la clase mici către clase mari sau invers. Se repetă pașii până când media intră în interval. În practică diferențele sunt mici. Fluxurile sunt corelate în măsurătorile discutate. Fundamentarea folosește idei din teoria distribuțiilor ponderate [183] [184] și din literatura despre calibrarea eșantioanelor [185].

4.4.3. Estimarea repartițiilor prin minimizarea entropiei relative

Metoda tratează determinarea repartiției dimensiunilor de bloc la nivelul NFS ca pe o problemă de optimizare informațională. Se pornește de la clasele de dimensiuni de bloc. Pentru fiecare clasă există o distribuție observată la hipervizoare. La NFS este cunoscută media dimensiunii blocului exprimată ca octeți pe operație. Scopul este să obținem o repartiție pe aceleași clase care este cât mai apropiată de distribuția de la hipervizoare în sens informațional [186]. În același timp repartiția trebuie să respecte două condiții. Suma probabilităților este unu. Media sub această repartiție este egală cu media măsurată la NFS. Toate ponderile sunt nenegative.

Teoria transmiterii informației oferă o soluție standard pentru astfel de probleme [186]. Repartiția căutată se obține printr-o reponderare a distribuției de la hipervizoare cu un factor care crește sau scade în funcție de dimensiunea fiecărei clase. Intensitatea acestei reponderări este controlată de un singur parametru real numit aici λ . Pentru orice valoare a lui λ rezultă o repartiție validă. Există și un factor de normalizare care asigură că suma probabilităților rămâne unu. Valoarea corectă a lui λ este cea pentru care media sub repartiția reponderată devine exact media măsurată la NFS. Astfel se obține o distribuție calibrată care păstrează forma de bază de la hipervizoare și respectă constrângerea mediei la NFS.

Proprietățile matematice ale acestei construcții asigură existență și unicitate. Se definește o funcție ajutătoare egală cu logaritmul sumei ponderate cu exponențiale. Această funcție este convexă [187]. Derivata ei este chiar media dimensiunii blocului sub repartiția reponderată. Această derivată crește strict atunci când λ crește. Când λ tinde spre minus infinit media se apropie de cea mai mică dimensiune din clasă. Când λ tinde spre plus infinit media se apropie de cea mai mare dimensiune din clasă. Deci pentru orice medie aflată între limita inferioară și limita superioară există o singură valoare a lui λ care rezolvă problema.

Determinarea numerică a lui λ se face simplu prin căutare binară [188]. Mai întâi se calculează media obținută cu λ egal cu zero. Dacă această valoare este egală cu media de la NFS atunci repartiția NFS coincide cu repartiția de la hipervizoare. Dacă media dorită este mai mare se alege un capăt superior pozitiv pentru care media depășește ținta. Capătul inferior rămâne zero. Dacă media dorită este mai mică se alege un capăt inferior negativ pentru care media scade sub țintă. Capătul superior rămâne zero. Apoi se împarte intervalul în două la fiecare pas. Se evaluează media în punctul de mijloc. Dacă media este peste țintă se mută capătul superior în mijloc. Dacă media este sub țintă se mută capătul inferior în mijloc. Procedura continuă până când eroarea absolută față de media dorită scade sub un prag ales de la început. Valoarea finală a lui λ dă repartiția calibrată. Aceasta devine distribuția pe clase de dimensiuni a blocurilor la nivelul serverului NFS.

4.5. Metodologie pentru evaluarea performanțelor sistemelor de stocare

Evaluarea performanțelor unui sistem de stocare partajat poate fi abordată din multiple direcții, deoarece arhitectura sistemelor de stocare permite, de obicei, o varietate considerabilă de combinații. În momentul în care sistemul de stocare dispune de mai multe discuri, modul de organizare a spațiului fizic reprezintă prima alegere critică. La fel de importante sunt și deciziile legate de interfețele de rețea și de modul de partajare a resurselor. Într-un scenariu modern, alegerea între TCP și RDMA, respectiv alegerea între diferite modalități de acces concurent la volumele partajate, determină latențele, stabilitatea și gradul de încărcare a procesorului gazdă.

4.5.1. Configurații ale sistemelor de stocare partajate

Analiza pornește de la felul în care se configurează stocarea partajată și de la modul de evaluare. Variabilele principale sunt organizarea discurilor prin RAID0 sau RAID10 sau RAID5 și dimensiunea segmentului distribuit și sistemul de fișiere și protocoalele de rețea și modul de partajare a resurselor. Testele sintetice nu ajung. Metodologia folosește profiluri de intrare ieșire obținute din observații reale. Profilurile includ dimensiunile frecvente ale blocurilor și raportul IOPS între direcții și raportul volumelor dintre citiri și scrieri. Scopul este alegerea unei combinații care echilibrează performanța și latența și stabilitatea în condiții apropiate de utilizare.

Abordarea permite caracterizare detaliată și comparație echitabilă. Factorul de paralelizare se definește ca produsul dintre numărul de fire de execuție și adâncimea cozii de intrare ieșire. Intervalul testat ajunge până la 1024. Dincolo de acest prag percentila p99 a latenței crește puternic iar uzura mediilor crește. Fiecare test durează 60 de secunde. Există o perioadă de încălzire de 10 secunde. Criteriul principal este p99 al latenței cu prag 5 ms.

Configurațiile sunt organizate pe nivel RAID și pe dimensiunea segmentului și pe sistemul de fișiere cu modul de partajare. RAID0 maximizează debitul fără redundanță. RAID10 combină distribuția cu oglindirea. RAID5 folosește paritate și spațiu eficient. Dimensiunile de segment testate sunt 64 KB și 128 KB și 256 KB și 512 KB și 1024 KB și 2048 KB. Variantele cu NFS peste RDMA folosesc ext4 sau XFS sau ZFS. Ext4 oferă stabilitate și compatibilitate. XFS este potrivit pentru volume mari și acces paralel. ZFS adaugă verificare a integrității și cache avansat. Varianta cu iSCSI peste RDMA folosește OCFS2 pentru acces concurent la nivel de bloc. Interpretarea rezultatelor ține cont și de nivelul la care are loc partajarea.

4.5.2. Definirea mărimilor măsurabile

Testele au rulat cu fio în mod distribuit client server. Un server principal a orchestrat execuția. A trimis fișierele de job către procesele fio server de pe fiecare client și a colectat rezultatele în format JSON. Mecanismul a menținut condiții omogene și a permis aplicarea unitară a factorului de paralelizare pe tot ansamblul. Toate testele au rulat pe toți clienții. Încărcarea s-a distribuit uniform pe serverele de test și pe infrastructura de rețea. Configurarea a ținut cont că există 8 clienți. Minimul de paralelism vizibil la server este 8 atunci când fiecare client rulează cu un singur fir și adâncime a cozii egală cu 1.

Structurarea a avut două etape. În prima etapă fiecare combinație de RAID și dimensiune de segment și sistem de fișiere a fost evaluată separat în funcție de factorul de paralelizare. Graficele au urmărit evoluția debitului și a IOPS și a latențelor. În a doua etapă rezultatele au fost grupate pe seturi. Astfel au devenit posibile comparațiile între dimensiunile segmentelor și extragerea configurațiilor considerate optime. Pentru acest scop au fost folosite grafice comparative care sintetizează și ordonează valorile. Vizualizările reflectă nivelul detaliu pe test și nivelul comparație pe set.

4.5.2.1. Diagrame individuale per test

În prima etapă au fost construite seturi de diagrame pentru fiecare dimensiune a segmentului distribuit din RAID. Explicațiile folosesc exemple dintr-un test real pentru a clarifica mărimile și valorile ilustrate.

Primul grafic arată lățimea de bandă medie agregată la citire și la scriere în funcție de factorul de paralelism. Sunt evidențiate doar configurațiile unde p99 rămâne sub cinci milisecunde. Punctele accentuate marchează zona considerată acceptabilă din perspectiva latenței. Punctele estompate din fundal arată restul măsurătorilor. Se vede direct ce debit mediu se obține fără depășirea limitei de latență și care este intervalul de paralelism în care sistemul rămâne utilizabil.

Diagrama următoare urmărește p95 și p99 în funcție de factorul de paralelism. Dacă cele două curbe stau aproape atunci distribuția latențelor este compactă și sistemul se menține previzibil. Dacă p99 se îndepărtează repede de p95 odată cu creșterea paralelismului atunci distribuția se lărgeste și apare imprevizibilitate. Această vedere indică și cât paralelism poate susține subsistemul.

Un alt grafic arată lățimea de bandă medie și IOPS în funcție de paralelism. La început creșterea este aproape proporțională deci resursele suplimentare sunt folosite eficient. După atingerea saturației curba se aplatizează și nu mai apar câștiguri fără penalizări de latență. Analizat singur acest grafic poate induce ideea că variantele cu valori absolute mai mari sunt mai bune. Corelarea cu graficele de latență arată însă că un debit mai mare obținut cu p99 în creștere accentuată nu aduce un avantaj practic.

O diagrama pune în relație lățimea de bandă totală și p99 pe scară logaritmă. Reprezentarea lasă să se vadă fin diferențele din zona sensibilă unde câteva milisecunde schimbă percepția. Configurațiile cu debit mare și p99 scăzut se disting rapid. Cele care obțin debit ridicat cu prețul unei latențe mari se văd la fel de ușor. Scala logaritmă nu ascunde variațiile mici dar relevante și face clar cum scade performanța percepută când sistemul este împins spre limită.

O altă diagrama arată IOPS în raport cu p99 pe scară logaritmă. Se vede momentul în care creșterea IOPS nu mai compensează degradarea latenței. Zona utilă de funcționare se poate delimita cu precizie.

Ultimul grafic descrie câștigul de lățime de bandă atunci când paralelismul se dublează. Câștigul este raportul dintre debitul obținut la un anumit nivel de paralelism și debitul obținut la jumătate din acel nivel. În zona aproape liniară câștigul este aproape unu ceea ce arată folosirea eficientă a resurselor adăugate. Aproape de saturație câștigul scade sub unu uneori brusc. Fiecare dublare aduce tot mai puțin. Graficul arată clar cât mai aduce fiecare dublare a paralelismului.

Aceste diagrame au valoare împreună. Lățimea de bandă în funcție de paralelism arată scalarea dar poate induce interpretări greșite fără diagrama de latențe. Câștigul la dublare arată eficiența din punctul de vedere al debitului dar nu descrie consistența latenței. Diagrama

cu testele eligibile după latență completează această lipsă. Privite împreună ele arată limitele și zonele de funcționare care merită folosite.

4.5.2.2. Diagrame comparative

După analiza fiecărei combinații la nivel de test au fost construite diagrame comparative pe seturi. Scopul este evidențierea efectului dimensiunii segmentului din RAID pentru o configurație dată și păstrarea condiției p99 sub 5 ms.

Prima reprezentare selectează pentru fiecare granularitate punctul cel mai bun cu debit total și cu IOPS maxim sub pragul de latență. Astfel se obține o ierarhie inițială a dimensiunilor de segment. O altă diagramă arată evoluția debitului mediu total în funcție de factorul de paralelizare pentru toate granularitățile și marchează separat punctele care trec peste pragul p99. Se vede dacă un anumit segment menține debite ridicate pe o plajă largă sau dacă performanța cade devreme. O a treia diagramă urmărește IOPS în funcție de paralelizare pentru toate granularitățile și pune în evidență compromisul dintre debit și numărul de operații. Urmează o frontieră debit față de p99 pe scară logaritmică care face vizibile diferențele în zona valorilor mici ale latenței și arată direct schimbul dintre viteză și răspuns previzibil. Un grafic separat reține primele trei rezultate sub 5 ms pentru decizie rapidă.

Valoarea apare când aceste diagrame sunt citite împreună. Bara globală oferă vârful fiecărei granularități. Curbele pe paralelizare arată stabilitatea. Frontiera debit față de p99 explică schimbul dintre performanță și latență. Selecția cu primele trei puncte concentrează opțiunile utile.

Abordarea se diferențiază de testele clasice cu scenarii fixe. Metoda explorează factorul de paralelizare până la 1024. Corelează debitul și IOPS cu distribuția latențelor. Folosește p99 cu prag de 5 ms ca filtru de relevanță. Integrează rezultatele prin grafice complementare astfel încât analiza să rămână aplicabilă în practică.

4.6. Concluzii intermediare

Se definește o metodologie pentru identificarea tiparelor de intrare ieșire în platforme de virtualizare. Integrarea cu orchestratorul se face prin hook uri pentru etapele deploy și boot. Evenimentele sunt trimise către un server de monitorizare cu indicarea nodului gazdă. Legarea acestor evenimente de seriile de date permite corelarea exactă a stărilor mașinilor virtuale cu măsurătorile.

Profilul de performanță urmărește debitul IOPS latențele raportul citire scriere și distribuția dimensiunilor de bloc. Măsurarea are două puncte. La nivelul hipervizoarelor pentru a surprinde cererile generate de mașinile virtuale. La nivelul serverului de stocare în memorie volatilă pentru a surprinde activitatea după cache după stiva software și după protocolul de transport.

Metoda pornește de pe un disc virtual în RAM. Apoi urmează testarea pe sisteme reale de stocare. Se compară niveluri RAID sisteme de fișiere soluții integrate și acces concurent pe același volum printr un sistem de fișiere de cluster. Accentul cade pe comportamentul raportat la profilul obținut și nu pe maxime sintetice. Sunt analizate și diferențele dintre formatele de disc ale mașinilor virtuale.

Configurațiile includ imagini Linux și Windows distribuite uniform pe noduri. Scripturile interne execută instalări actualizări inițializări de servicii și date. Execuțiile au două moduri. Un mod sincron cu pornire simultană. Un mod etapizat cu întârzieri între instanțe. Seriile de test combină modurile de execuție cu formatele de disc. Seturile sunt omogene Linux Windows sau mixt. Fluxul operațional este unic. Se pornește monitorizarea pe hipervizoare și pe server. Se orchestrează lansarea și urmărirea execuțiilor. Se face oprirea controlată și curățarea resurselor. Apoi se agregă și se procesează datele pentru grafice și fișiere de analiză.

Metricile includ marcaje temporale din ciclul de viață ale mașinilor virtuale. Includ seriile de debit IOPS latențe și distribuții ale dimensiunilor de bloc la nivelul hipervizoarelor. Includ debit și IOPS la nivelul serverului de stocare. Se adaugă indicatori de utilizare a resurselor pe noduri. Rezultatele sunt prezentate prin diagrame care verifică distribuția sarcinii pe noduri duratele stărilor evoluțiile de debit și IOPS rapoartele citire scriere hărțile temporale ale dimensiunilor de bloc și cronologiile pe rulare.

Capitolul V - Analiza rezultatelor experimentale

5.1. Validarea preliminară a infrastructurii de monitorizare

Pentru a valida funcționarea corectă a infrastructurii și a sistemelor de monitorizare a fost efectuat un test preliminar folosind o distribuție uniformă pe cele 8 noduri de găzduire a 240 de mașini virtuale. Acestea au rulat Linux plecând de la o imagine de tip QCOW2 și ca sarcină de lucru a fost aleasă metoda de testare cu sincronizare globală și fără pauze în timpul execuției, și anume un scenariu de încărcare maximă.

5.1.1. Test pentru validarea preliminară

Sunt definite două ferestre temporale pentru validare. Prima numită `deploy+boot` acoperă intervalul de la instanțiere până la ultima raportare `ready` și durează aproximativ 34 s. Aceasta include crearea instanțelor și atingerea stării de funcționare a sistemelor de operare. A doua fereastră merge de la primul start la ultimul end și durează aproximativ 555 s. În acest interval rulează sarcina de lucru și se produce trafic de intrare ieșire susținut. Delimitările separă faza cu vârfuri tranzitorii de faza cu regim mai stabil potrivită pentru extragerea unui profil repetabil.

Corelarea în timp între seriile raportate de mașinile virtuale și cele măsurate pe `tmpfs` confirmă suprapunerea dinamicii traficului. În fereastra sarcinii coeficientul Pearson [189] este $p = 0.934$ la citire și $p = 0.989$ la scriere pentru lățimea de bandă. Nu apar întârzieri detectabile ceea ce arată că sincronizarea ceasurilor este corectă. Pentru IOPS valorile sunt $p = 0.985$ la citire și $p = 0.995$ la scriere. Curbele se aliniază secundă cu secundă conform figurilor 66 și 67 iar diferențele provin din particularități ale mecanismelor de citire și scriere.

Analiza volumelor în fereastra sarcinii arată la citire o creștere de 14 punct 28 la sută aproximativ 651 GB față de 570 GB și la scriere o creștere de 9 punct 55 la sută aproximativ 2.075 GB față de 2.274 GB la nivelul `tmpfs`. Structura cererilor observată la hipervizoare completează tabloul. La citire sunt dominante blocurile de 128 KB cu 34 punct 66 la sută urmate de 64 KB cu 29 punct 23 la sută și 32 KB cu 22 punct 44 la sută. La scriere apar 32 KB cu 21 punct 07 la sută apoi 64 KB cu 18 punct 74 la sută și 256 KB cu 14 punct 64 la sută.

Comparând măsurătorile de la hipervizoare cu cele de la NFS se disting două tipuri de discrepanțe. Unele sunt integrale și se văd în valorile cumulate pe toată fereastra. Altele sunt instantanee și apar la scara secundelor când pot exista inversări temporare ale raportului între volumele raportate de cele două surse.

5.1.2. Diferențele integrale ale volumelor transferate

Secțiunea explică diferențele integrale ale volumelor măsurate. Ele apar din modul de lucru al formatului QCOW2 și din stiva de stocare NFS peste RDMA cu fișiere pe `tmpfs`. La scriere serverul a raportat un plus de aproximativ 9 punct 55 la sută. Cauzele probabile sunt actualizările tabelelor L1 și L2 din QCOW2 [176]. Actualizarea tabelului de referințe. Alocarea inițială a clusterelor cu umplere cu zerouri. Granularitatea pe pagini de memorie de patru kilobytes în `tmpfs` și în NFS care poate contabiliza o pagină întreagă la o scriere parțială. Aceste efecte apar în măsurătorile de pe `tmpfs` deoarece reflectă accesul real asupra fișierului QCOW2 și nu doar cererea brută a mașinilor virtuale.

La citire discrepanța este mai mare. NFS a raportat aproximativ 14 punct 28 la sută în plus. Motivele sunt accesarea metadatelor L1 și L2 când nu se află în cache. Readahead și prefetch ale clientului NFS care pot aduce mai multe date decât a solicitat instanța. Granularitatea pe pagini care poate duce la citirea unei pagini întregi pentru o cerere mai mică. Agregarea cererilor adiacente de către QEMU sau de către NFS ceea ce transformă mai multe citiri mici într-una mai mare. Porțiunile suplimentare sunt transferate și apar în contorul NFS dar nu apar la nivelul hipervizoarelor.

Dimensiunea medie a blocului rămâne apropiată între surse. La citire aproximativ 37 punct 92 kilobytes pe operație la nivelul mașinilor virtuale și aproximativ 37 punct 14 kilobytes pe operație la nivelul `tmpfs`. La scriere aproximativ 48 punct 5 kilobytes pe operație la nivelul

mașinilor virtuale și aproximativ 47 punct 4 kilobytes pe operație la nivelul tmpfs. Rezultă că efectele descrise schimbă mai ales numărul de operații și volumul total iar forma cererii tipice rămâne similară.

5.1.3. Diferențele instantanee ale volumelor transferate

Secțiunea descrie diferențele instantanee ale volumelor transferate. Pe întreaga fereastră NFS raportează volume mai mari decât hipervizoarele. La rezoluție de o secundă apar însă intervale scurte în care debitul cumulat de la hipervizoare depășește temporar debitul NFS mai ales la citire. Explicațiile posibile sunt mai multe. Ferestrele de măsurare au pas de o secundă și o operație mare poate începe la finalul unei ferestre și se poate încheia în următoarea astfel împărțirea volumului diferă între surse și apar discrepanțe locale. Datele pot fi deja în cache în QEMU sau în nodul de găzduire astfel livrarea către mașina virtuală nu creează trafic nou pe NFS în acel moment iar contorul hipervizoarelor crește imediat. Readahead poate aduce date în avans astfel contorul NFS se modifică înainte ca hipervizoarele să raporteze cererea. Prefetch și citirile speculative pot solicita date suplimentare înainte de cererea reală și apar vârfuri pe serverul NFS fără corespondent imediat la nivelul hipervizoarelor. Aceste efecte sunt punctuale iar tendința globală rămâne neschimbată. Pentru un profil de intrare ieșire corect aceste inversări trebuie documentate iar analiza trebuie să separe decalajele temporale de efectele interne ale formatului QCOW2 și ale stivei NFS.

5.1.4. Concluzii intermediare privind testul de validare

Validarea pe două puncte de monitorizare arată că aceleași operații sunt văzute coerent din unghiuri diferite. NFS cu tmpfs măsoară volumul efectiv transferat și proporția citire scriere și include traficul produs de mecanisme interne precum metadata. Hipervizoarele descriu structura cererilor prin dimensiunile blocurilor și repartitia lor pe clase. În fereastra de sarcină seriile sunt puternic corelate. Diferențele integrate pe întreaga perioadă sunt moderate și au același sens. NFS raportează volume mai mari atât la citire cât și la scriere

Din aceste observații rezultă un mod de construire a profilului de intrare ieșire. Volumul total și raportul citire scriere se iau din măsurarea NFS cu tmpfs. Distribuția dimensiunilor de bloc se ia din măsurarea pe hipervizoare pentru fiecare mașină virtuală. Repartitia pe clase se ponderă apoi cu raportul dintre volumele NFS și volumele văzute pe hipervizoare separat pe citire și pe scriere. Astfel forma distribuției rămâne fidelă cererilor emise de clienți iar scara volumelor reflectă traficul real din rețea și pe server.

Această combinație elimină erorile care apar dacă se folosește o singură sursă. Doar hipervizoarele ar subestima încărcarea pe stocare. Doar NFS ar pierde fidelitatea structurii cererilor. Concluzia este că alăturarea volumelor și ponderilor din NFS cu distribuțiile de bloc din hipervizoare oferă un profil de intrare ieșire echilibrat și aplicabil direct în testele comparative pe alte sisteme de stocare

5.2. Determinarea profilurilor de intrare/ieșire

Pentru definirea profilurilor de intrare ieșire au fost rulate toate seriile și toate seturile. Fiecare set a avut trei rulări și rezultatele au fost mediate neponderat. Pentru fiecare set au fost generate fișiere de sinteză. Acestea includ histograme ale dimensiunilor de bloc observate la nivelul mașinilor virtuale pe citire și pe scriere. Mai includ valorile medii ale lățimii de bandă la NFS în fereastra stabilă și valorile medii ale numărului de operații pe secundă în aceeași fereastră.

Dimensiunile de bloc au fost încadrate într-o formă canonică cu praguri de la 512B la 2MB. Mediile pe operație au fost exprimate în MB pe operație și provin din raportul dintre lățime de bandă și număr de operații. Profilurile au fost construite prin proiectarea distribuției empirice a blocurilor măsurată pe mașini virtuale pe fiecare direcție și pentru fiecare set către o distribuție care respectă media țintă impusă de măsurătorile NFS. Pentru fiecare set distribuțiile obținute pe rulări au fost apoi mediate neponderat și renormalizate. Rezultatul este

profilul final pe citire și pe scriere. Proportțiile citire/scriere au fost calculate din mediile neponderate ale numerelor de operații.

Pentru repartițiile blocurilor la nivelul NFS a fost folosită metoda de minimizare a entropiei relative prezentată anterior.

5.2.1. Scenarii de încărcare medie

5.2.1.1. Test cu imaginile discurilor virtuale în format QCOW2

Pentru rularea acestei serii de teste a fost folosită următoarea configurație:

- Pentru toate seturile de teste, configurațiile comune au fost următoarele:
 - Imaginile discurilor virtuale au fost toate în format QCOW2;
 - Planificatorul a fost setat cu parametri normali, pentru o distribuție uniformă a instanțelor virtuale, cu o execuție a ciclurilor de planificare la 1 secundă, cu 16 fire de execuție pentru procesele de planificare, monitorizare și alocare de resurse;
 - Sarcina de lucru a fost configurată cu întârzieri aleatorii între activitățile programate, în intervalul [5,60] de secunde, indiferent de sistemul de operare folosit;
 - Pornirea sarcinilor de lucru nu a fost sincronizată, fiecare mașină virtuală a lansat în execuție sarcina după un interval de timp aleatoriu [5,40] secunde.
- Au fost executate următoarele seturi de teste, definite astfel:
 1. Linux - de la 16 la 320 de mașini virtuale (în pași de 16), rezultând 20 de rulări;
 2. Windows - de la 16 la 160 de mașini virtuale (în pași de 16), rezultând 10 rulări;
 3. Mixt (Linux + Windows) - de la 16 (8+8) la 256 (128+128) de mașini virtuale (în pași de 16), rezultând 14 rulări.

Profilurile de intrare/ieșire rezultate sunt ilustrate în tabelele din lucrare.

5.2.1.2. Test cu imagine disc RAW

Pentru rularea acestei serii de teste a fost folosită următoarea configurație:

- Pentru toate seturile de teste, configurațiile comune au fost următoarele:
 - Imaginile discurilor virtuale au fost toate în format RAW;
 - Planificatorul a fost setat cu parametri normali, pentru o distribuție uniformă a instanțelor virtuale, cu o execuție a ciclurilor de planificare la 1 secundă, cu 16 fire de execuție pentru procesele de planificare și alocare de resurse;
 - Sarcina de lucru a fost configurată cu întârzieri aleatorii între activitățile programate, în intervalul [5,60] de secunde, pentru toate sistemele de operare;
 - Pornirea sarcinilor de lucru nu a fost sincronizată, fiecare mașină virtuală a lansat în execuție sarcina după un interval de timp aleatoriu [5,40] secunde.
- Au fost executate următoarele seturi de teste, definite astfel:
 1. Linux - de la 16 la 256 de mașini virtuale (în pași de 16), rezultând 16 rulări;
 2. Windows - de la 16 la 80 de mașini virtuale (în pași de 16), rezultând 5 rulări;
 3. Mixt (Linux + Windows) - de la 16 (8+8) la 128 (64+64) de mașini virtuale (în pași de 16), rezultând 8 rulări.

Profilurile de intrare/ieșire rezultate sunt ilustrate în tabele în lucrare.

5.2.2. Scenarii de încărcare maximă

5.2.2.1. Test cu imagine disc QCOW2

Pentru rularea acestei serii de teste a fost folosită următoarea configurație:

- Pentru toate seturile de teste, configurațiile comune au fost următoarele:
 - Imaginile discurilor virtuale au fost toate în format QCOW2;
 - Planificatorul a fost setat pentru o instanțiere cât mai rapidă a mașinilor virtuale cu ajutorul scripturilor specializate prin apeluri XML-RPC;

- Sarcina de lucru a fost configurată fără întârzieri aleatorii între activitățile programate, cu execuție imediată, indiferent de sistemul de operare folosit;
- Pornirea sarcinilor de lucru a fost sincronizată, toate mașinile virtuale au lansat în execuție sarcina imediat după raportarea.
- Au fost executate următoarele seturi de teste, definite astfel:
 1. Linux - de la 16 la 320 de mașini virtuale (în pași de 16), rezultând 20 de rulări;
 2. Windows - de la 16 la 160 de mașini virtuale (în pași de 16), rezultând 10 rulări;
 3. Mixt (Linux + Windows) - de la 16 (8+8) la 224 (112+112) de mașini virtuale, rezultând 13 rulări.

Profilurile de intrare/ieșire rezultate sunt descrise în table în lucrare.

5.2.2.2. Test cu imagine disc RAW

Pentru rularea acestei serii de teste a fost folosită următoarea configurație:

- Pentru toate seturile de teste, configurațiile comune au fost următoarele:
 - Imaginile discurilor virtuale au fost toate în format RAW;
 - Planificatorul a fost setat pentru o instanțiere cât mai rapidă a mașinilor virtuale cu ajutorul scripturilor specializate prin apeluri XML-RPC;
 - Sarcina de lucru a fost configurată cu întârzieri aleatorii între activitățile programate, în intervalul [5,60] de secunde, în toate sistemele de operare;
 - Pornirea sarcinilor de lucru nu a fost sincronizată, fiecare mașină virtuală a lansat în execuție sarcina după un interval de timp aleatoriu [5,40] secunde.
- Au fost executate următoarele seturi de teste, definite astfel:
 1. Linux - de la 16 la 256 de mașini virtuale (în pași de 16), rezultând 16 de rulări;
 2. Windows - de la 16 la 80 de mașini virtuale (în pași de 16), rezultând 5 rulări;
 3. Mixt (Linux + Windows) - de la 16 (8+8) la 128 (64+64) de mașini virtuale, rezultând 8 rulări.

Profilurile de intrare/ieșire rezultate sunt ilustrate în tabele în lucrare.

5.2.3. Analiza profilurilor de intrare/ieșire determinate

5.2.3.1. Efectul formatului imaginii de disc: QCOW2 vs. RAW

În regim mixt cu încărcare medie trecerea de la QCOW2 la RAW schimbă balanța lățimii de bandă la NFS în favoarea scrierilor. Pentru QCOW2 citirea are aproximativ 17.2% iar scrierea 82.8%. Pentru RAW citirea coboară spre 6.5% iar scrierea urcă la 93.5%. Diferența se vede și în dimensiunile blocurilor. La QCOW2 citirile se adună în zona 32 până la 128 KB iar scrierile sunt răspândite între 16 și 64 KB cu o prezență modestă la 512 KB aproape 5%. La RAW scrierile de 512 KB devin dominante aproape 36% ceea ce sugerează agregare către blocuri mari pe calea de scriere. În încărcare maximă comportamentul se schimbă. La QCOW2 ponderea scrierilor de 512 KB rămâne ridicată aproape 36%. La RAW scrierile migrează către 16 până la 64 KB iar 512 KB scade spre 3%. Rezultatul arată că formatul imaginii și nivelul de încărcare modelează structura traficului. La mediu RAW favorizează blocuri mari la scriere. La maxim QCOW2 păstrează 512 KB ridicat iar RAW se împărțește pe blocuri medii.

Compararea numerelor de operații cu lățimea de bandă confirmă diferențele. În seturile mixte cu încărcare medie QCOW2 arată la NFS aproximativ 35.9% citire și 64.1% scriere la IOPS iar la hipervizoare aproximativ 28.6% citire și 71.4% scriere. NFS vede astfel relativ mai multe citiri fapt compatibil cu efecte de cache și readahead. Pentru RAW IOPS la NFS rămâne orientat spre scriere aproape 62.9% iar lățimea de bandă trece net spre scrieri aproape 93.5%.

Mecanismele interne explică aceste rezultate. QCOW2 adaugă trafic pe metadate prin tabelele de adresare și refcount și prin alocări de clustere. RAW elimină acest cost și poate schimba modul în care hipervizorul și lanțul NFS RDMA agregă sau fragmentează blocurile în funcție de regimul de lucru.

5.2.3.2. Diferențe între sisteme de operare: Linux vs. Windows vs. Mixt

În Windows la NFS numărul de operații pe secundă tinde spre echilibru cu aproximativ 52.3 la citire și 47.7 la scriere. Lățimea de bandă rămâne orientată spre scriere cu aproximativ

83. Din distribuția pe blocuri la citire se vede un vârf la 512 B în jur de 51 urmat de 32 KB în jur de 19. Rezultă multe citiri foarte mici. În scenariul mixt apar mai puține citiri ca număr la NFS. Windows aduce mai multe tranzacții scurte la citire. Lățimea de bandă rămâne tot spre scriere.

În mixt la NFS IOPS sunt aproximativ 35.9 la citire și 64.1 la scriere iar lățimea de bandă aproximativ 17.2 la citire și 82.8 la scriere. La hipervizoare IOPS sunt aproximativ 28.6 la citire și 71.4 la scriere iar lățimea de bandă aproximativ 16.0 la citire și 84.0 la scriere.

La încărcare maximă diferențele cresc. În mixt scrierea domină în aproape toate mărimile iar clasa 512 KB devine principală la scriere cu QCOW2. Se vede tendință de consolidare în blocuri mari. Cu RAW scrierile se mută către 16 până la 64 KB iar 512 KB scade mult. În Windows la hipervizoare scrierea are vârfuri la 16 KB și 64 KB cu un front larg între 16 KB și 128 KB. Profilul rămâne orientat spre scriere dar este mai fragmentat decât în mixt cu QCOW2.

În ansamblu la NFS ponderea pe direcții este mai mult spre scriere decât la hipervizoare în majoritatea seturilor. La citire NFS raportează adesea mai multe operații decât hipervizoarele în aceeași fereastră. Explicația ține de cache și de readahead. O parte din citiri se livrează în avans pe server. Alteori datele sunt deja în cache local și se livrează fără trafic nou prin mecanisme zero copy în RDMA. La scriere toate operațiile traversează stiva și se contabilizează integral la NFS. QCOW2 adaugă trafic pe metadata prin tabele L1 și L2 și prin refcount precum și prin inițializări de clustere. Volumele integrate la NFS ies mai mari decât la hipervizoare. Acest efect apare constant ca sens. Nu indică o eroare. Hipervizoarele surprind forma cererilor pe dimensiuni de bloc. Această informație lipsește la NFS dar este utilă pentru a înțelege granularitatea reală a accesului.

5.3. Investigații experimentale asupra sistemelor partajate de stocare

5.3.1. Definirea cadrului experimental

Secțiunea definește cadrul pentru evaluarea unor sisteme de stocare cu unități NVMe. Scopul este configurarea unui server de stocare partajată care să deservească o platformă de virtualizare. Discurile virtuale ale mașinilor vor fi găzduite pe acest server. Testele folosesc fio și un profil de acces extras din scenariul de utilizare educațional. Profilul este mixt Linux și Windows. Imaginile de disc sunt QCOW2.

Infrastructura de test este apropiată de cea folosită la profilare. Serverul are două procesoare cu câte 24 de nuclee. Are 512 GB memorie. Are 12 unități NVMe. Are două porturi de rețea cu 100 Gbps fiecare. Acest server rulează rolul de stocare. Pe partea de clienți sunt opt noduri identice cu aceeași conectivitate. Directorul partajat este exportat prin NFS peste RDMA. Configurațiile de rețea și de sistem rămân aceleași.

Fio rulează în mod client server. Pe fiecare nod client rulează fio server. Controlul este lansat de pe serverul de stocare prin fio client. Nu există un nod separat pentru orchestrare. Procesul de control așteaptă finalizarea testelor și primește rezultatele agregate. Lansarea este sincronizată în timp pe toate nodurile.

Utilizarea procesoarelor și a memoriei este monitorizată permanent pe toate sistemele. Se folosește collectl cu o granularitate de o secundă. Vor fi testate patru configurații de sisteme de fișiere în combinație cu mai multe variante RAID. Fiecare variantă este analizată separat. Vor fi prezentate numai diagramele comparative cele mai relevante. Restul graficelor se află în anexe.

5.3.2. Sistem de fișiere ext4 cu RAID software - configurare și rezultate

Sistemul de fișiere ext4 a fost testat peste o matrice RAID software cu douăsprezece unități NVMe cu partajare prin NFS peste RDMA. Testele au rulat cu fio în mod client server cu inițiere control și colectare centralizate și sincronizate pe toate nodurile. Configurarea ext4 a respectat alinierea la segmentarea RAID pentru fiecare scenariu. Parametrii stride și stripe width au fost ajustați la fiecare pas după dimensiunea segmentului RAID pentru a evita accesări parțiale și pentru a reduce penalitățile la scriere. Această abordare permite ca

diferențele observate să fie atribuite nivelului RAID și granularității segmentului și nu efectelor de nealiniere.

5.3.2.1. RAID0 + ext4 - rezultate experimentale

Analiza arată diferențe între configurații determinate de dimensiunea segmentului distribuit și de factorul de paralelizare. La FP mic performanța crește previzibil iar variațiile sunt reduse. De la FP mediu curbele se separă. Segmentele mari rămân eligibile pe o durată mai lungă și păstrează valori ridicate de debit. Configurațiile medii își restrâng treptat domeniul valid. Segmentele mici își pierd eligibilitatea devreme. Segmentul de 256 KB are un comportament inegal și o zonă îngustă de eligibilitate.

Distribuția punctelor valide diferențiază clar configurațiile. Segmentele mari păstrează densitate ridicată pe o plajă extinsă de FP și oferă stabilitate lângă zona de maxim. Configurațiile medii au grupări separate și întreruperi vizibile. Configurațiile cu segmente mici rămân active pe intervale scurte de FP și dispar rapid la creșterea încărcării. Segmentul de 256 KB confirmă un profil fragmentat fără o zonă de stabilitate comparabilă.

Sinteza celor mai bune puncte susține aceeași ierarhie. Segmentele mari se plasează constant în partea superioară prin debit ridicat și IOPS ridicat și prin menținerea latenței sub prag. Configurațiile medii concurează doar pe intervale înguste de FP iar diferența față de segmentele mari crește odată cu sarcina. Segmentul de 256 KB rămâne constant sub celelalte cu puține puncte valide și cu scăderi rapide după FP mediu. Segmentele mici pot fi competitive doar la FP redus și ies din zona optimă înainte de 512 KB.

În ansamblu se conturează două clase. Segmentele mari susțin paralelism ridicat pe un domeniu extins cu pierderi controlate aproape de saturație. Configurațiile medii rămân utile pe plaje înguste. Granularitățile mici se potrivesc doar la încărcare redusă. Segmentul de 256 KB rămâne neavantajos prin rezultate inconstante și prin zonă redusă de eligibilitate.

5.3.2.2. RAID10 + ext4 - rezultate experimentale

La FP mic curbele tuturor dimensiunilor de segment sunt apropiate și evoluția este previzibilă. Diferențele apar la FP mediu și ridicat. Segmentul de 512 KB menține cel mai mare debit și o zonă largă sub pragul de latență. Dimensiunea de 2 MB are o zonă mai îngustă și rămâne sub 512 KB. Dimensiunile intermediare funcționează în intervale bine delimitate. Dimensiunile mici rămân valabile doar la FP redus. Segmentul de 256 KB acoperă un domeniu larg de FP dar cu debite mai mici decât celelalte.

Distribuția punctelor eligibile confirmă aceste diferențe. La 512 KB apar multe puncte valide pe o plajă extinsă și trecerea spre depășirea pragului de latență este graduală. La 2 MB există puține puncte eligibile și performanța scade repede când FP crește. Dimensiunile intermediare păstrează puncte valide în porțiuni bine definite dar apar întreruperi în curbe. Dimensiunile mici au puncte aproape doar la FP redus și densitatea scade mult odată cu sarcina. La 256 KB acoperirea FP este mare dar debitul asociat rămâne scăzut.

Sinteza punctelor maxime valide dă o ordine clară. Pe primul loc 512 KB. Urmează 64 KB cu valori apropiate. Apoi 2 MB și 1024 KB ca grup intermediar. După acestea 128 KB. Pe ultimul loc 256 KB cu o distanță vizibilă. Corelarea dintre punctele de vârf și repartizarea lor pe curbe arată că 512 KB combină cel mai bine performanța și stabilitatea pe întreaga plajă de FP. Dimensiunea de 64 KB rămâne aproape dar pe un domeniu mai scurt. Dimensiunile de 2 MB și 1024 KB obțin rezultate medii cu FP valid limitat. Dimensiunea de 128 KB stă în aceeași zonă. La 256 KB numărul mare de FP valide nu este susținut de debite pe măsură. Setul RAID10 și ext4 evidențiază astfel această ierarhie prin nivelul maxim atins și prin lățimea domeniilor valide.

5.3.2.3. RAID5 + ext4 - rezultate experimentale

Testele pentru RAID5 cu ext4 arată un comportament diferit față de RAID0 și RAID10. În niciun scenariu nu apare vreun punct cu p99 sub 5 ms. Dimensiunea segmentului distribuit nu schimbă acest rezultat. Nici nivelul de paralelizare nu schimbă acest rezultat. Debitul poate atinge mii de MB pe secundă sau sute de mii de operații pe secundă. Aceste valori nu sunt

utile în lipsa unui control al latenței. Penalizarea introdusă de paritate nu poate fi ascunsă prin creșterea concurenței. Frontierele dintre debit și latență plasează toate punctele în zona p99 mare. Valorile se află între zeci și sute de milisecunde. Debitele rămân sub cele obținute cu RAID0 sau RAID10. Nu există compromis utilizabil între debit și timp de răspuns. Nu se poate stabili o ierarhie internă. Nu există dimensiune de segment optimă sub pragul de 5 ms. Nu există un top trei în bugetul de latență. Nu există o frontieră la stânga sus comparabilă cu celelalte configurații. Orice câștig de debit devine irelevant la aceste valori p99. Filtul p99 separă explicit punctele care ar produce blocaje reale în aplicații. RAID5 cu ext4 rămâne în afara domeniului util pentru această lucrare.

5.3.3. Sistem de fișiere XFS cu RAID software - configurare și rezultate

XFS a fost configurat în acord cu RAID. Dimensiunea segmentului din RAID a fixat valorile sunit și swidth din XFS. Scopul a fost limitarea penalizărilor de aliniere. Arhitectura XFS este gândită pentru sisteme mari și pentru paralelism ridicat. Jurnalizarea a rămas activă cu parametri adaptați. Frecvența de commit a fost redusă. Operațiile de jurnal au fost corelate cu segmentele RAID. Costul pe latență a scăzut. Montarea a folosit opțiuni orientate spre performanță cu păstrarea consistenței.

Testele s-au rulat cu fio în regim client server. Au acoperit seturi complete de valori pentru factorul de paralelizare. Validarea s-a făcut cu p99 ca prag de acceptare. Rezultatele arată nivelul obținut și felul în care XFS valorifică informația despre RAID pentru rezultate coerente sub pragul de latență.

5.3.3.1. RAID0 + XFS - rezultate experimentale

Analiza XFS pe RAID0 arată un comportament diferit față de ext4. Sub pragul percentila p99 egal 5 ms XFS menține debit mare și IOPS ridicat. Stabilitatea rămâne vizibilă și la valori mari ale factorului de paralelizare. Alte sisteme scad prin fragmentare sau prin salturi ale latenței.

Histograma arată lățimi de bandă totale în jur de 15 GB pe secundă. Apar și vârfuri peste acest nivel. IOPS total rămâne de regulă peste un milion pe combinațiile valide. Raportul dintre MB pe secundă și IOPS confirmă distribuția bună a cererilor prin jurnalizarea și alocarea pe extents.

Debitul mediu crește constant până la FP egal 64. Acolo apare maximul combinat. Dincolo de acest punct curbele nu mai cresc clar. Nici nu apar căderi bruște. Toate dimensiunile de segment se strâng într-o bandă îngustă cu valori între 12 și 14 GB pe secundă. Rezultatul arată că XFS folosește bine paralelismul RAID0 până la apropierea plafonului.

Câștigul la dublare este mare la FP mici. Între FP 16 și FP 8 se vede un raport de circa 1.6 până la 1.7. Câștigul scade spre 1 pe măsură ce paralelismul urcă. Concluzia este scalare bună la început și apoi intrare într-o zonă de plafon. Numărul de operații pe secundă trece de două milioane. Între FP 128 și FP 1024 apare un platou înalt. Resursele sunt folosite intens fără să apară degradări mari ale latenței. Cele mai bune puncte rămân sub 2 ms chiar la FP ridicat.

Ansamblul de rezultate indică o opțiune potrivită pentru scenarii care cer debit foarte mare și timp de răspuns controlat.

5.3.3.2. RAID10 + XFS - rezultate experimentale

XFS peste RAID10 păstrează un echilibru între debit și latență. Scalarea rămâne sub liniară pe măsură ce crește FP. Punctele valide se strâng într-o bandă cu debite între 8 și peste 11 GB pe secundă iar p99 între 0.5 și 3 ms. Rezultă o fereastră utilă în jur de FP 16 până la 64 iar uneori 128 unde creșterea paralelizării aduce câștig fără salturi ale timpilor de răspuns.

Curbele lățimii de bandă pentru dimensiuni diferite de segment se întâlnesc în zona FP 32 până la 64. Dincolo de acest punct apare un platou sau o alunecare lentă fără căderi bruște. Configurația este previzibilă dacă FP se așază în fereastra menționată.

Histograma cu debit și IOPS arată că scrierea domină cu peste două treimi din MB pe secundă conform profilului folosit. Citirea contribuie cu aproximativ 2 până la 2.7 GB pe

secundă în cele mai bune puncte. La IOPS scrierea ajunge la aproximativ 0.9 până la peste 1.0 milioane iar citirea rămâne în jur de 0.5 milioane. Performanța cea mai bună apare pentru segment distribuit de 64 KB.

5.3.3.3. RAID5 + XFS - rezultate experimentale

În combinația XFS cu RAID5 fereastra utilă de paralelism este îngustă. La FP 8 și la FP 16 apar aproape toate punctele cu p99 sub 5 ms. La FP de 32 sau mai mare p99 urcă rapid la zeci de milisecunde și peste. Debitul poate crește puțin dar latența iese din buget. Curbele lățimii medii de bandă filtrate la p99 sub 5 ms au doar două trepte. Se vede un spor de la FP 8 la FP 16 pentru unele granularități. Nu apare extindere la FP mai mari. Punctele gri arată multe măsurători la FP 32 și peste. Acestea nu intră în setul acceptat pe latență. Frontiera de latență taie testele cu FP mare, nu lipsa debitului.

În zona validă debitul total se află între 3.3 și 5.5 GB pe secundă în funcție de dimensiunea segmentului. Cele mai bune valori apar la FP 8 sau FP 16. Diferența între ele este moderată. La IOPS se vede aceeași imagine. Valorile sunt între 0.35 și 0.55 milioane pe secundă. Maximele apar în jur de 0.5 până la 0.55 milioane la FP 8. La FP 16 rămân în jur de 0.4 până la 0.42 milioane pentru punctele cu p99 sub 5 ms.

Frontiera debit față de p99 este limpede. Există un grup compact în stânga cu p99 sub 2 ms și debite între 3.3 și 4.6 GB pe secundă. Acest grup corespunde FP 8 și FP 16. Spre dreapta apare un nor dens cu p99 mult peste 5 ms. Debitele aici variază între 1.2 și 4.0 GB pe secundă. Aceste puncte nu respectă bugetul.

Histograma pe direcții explică contribuțiile. În cazurile de vârf cu p99 sub 5 ms scrierea domină debitul total cu aproximativ 2.9 până la 3.5 GB pe secundă dintr-un total de aproximativ 3.9 până la 5.5 GB pe secundă. Citirea adaugă aproximativ 0.9 până la 1.1 GB pe secundă. La IOPS scrierea se află în jur de 0.26 până la 0.35 milioane pe secundă. Citirea în jur de 0.12 până la 0.20 milioane. Profilul de test orientat spre scriere explică această asimetrie. Diferențele între granularități există dar nu schimbă concluzia. Valorile maxime sub buget apar la FP 8 și FP 16 indiferent de segmentul ales.

Concluzie operațională: pentru XFS cu RAID5 doar FP 8 până la 16 oferă simultan un debit acceptabil între 3.3 și 5.5 GB pe secundă un nivel de IOPS între 0.35 și 0.55 milioane și p99 sub 5 ms. Creșterea FP peste 16 degradează p99 în mod repetat chiar dacă debitul mai urcă puțin.

5.3.4. Sistem ZFS - configurare și rezultate

În testele cu ZFS au fost configurate vpool uri pentru a reflecta scenarii RAID prin mecanismele native ZFS. Echivalentul RAID0 a folosit un pool cu doisprezece vdev-uri independente. Fiecare vdev a corespuns unui SSD dedicat. Sarcina de citire și scriere s-a distribuit uniform pe toate dispozitivele. Configurația a reprodus modul de lucru al unui volum RAID0. Echivalentul RAID10 a folosit șase vdev-uri în oglindă. Fiecare vdev a utilizat două unități oglindite. Citirea s-a realizat în paralel din oricare copie. S-a menținut nivelul de redundanță specific RAID1. Echivalentul RAID5 a folosit un vdev de tip raidz1. Paritatea s-a distribuit pe toate unitățile. Configurația a oferit protecție la pierderea unui disc și o utilizare eficientă a spațiului.

Parametrii au fost uniformizați pentru comparabilitate între scenarii. Dimensiunea sectorului pe disc a fost setată cu ashift egal cu 12. Adresarea a folosit pași de 4 KB. Valoarea recordsize a variat între 64 KB și 2048 KB. Treptele au urmat aceleași valori ca în testele cu dimensiunea segmentului din RAID. Astfel comportamentul ZFS a fost observat pe întreaga gamă de granularități. De la tranzacții mici până la fluxuri secvențiale.

Pe durata testelor au rămas active toate mecanismele implicite ZFS. Nu s-a activat compresia. Nu s-a configurat slog dedicat. Nu s-a dezactivat jurnalizarea sincronă. S-a urmărit comportamentul nativ al sistemului în configurare standard. Scopul a fost observarea modului în care ZFS gestionează paralelismul și operațiile de intrare și ieșire în condiții aliniate cu celelalte sisteme de fișiere din studiu.

5.3.4.1. ZFS - 12 vdev (echivalent RAID0) - rezultate experimentale

Analiza începe cu evaluarea câștigului la dublarea factorului de paralelism. Graficul G al FP indică o scădere treptată a eficienței pe măsură ce FP crește. În zona FP mici de la 8 la 32 apare o creștere vizibilă. Rapoartele depășesc 1.2 și ajung uneori la 1.4 pentru blocuri de 64 KB și 128 KB. Rezultă că ZFS cu multe vdev active în paralel valorifică bine resursele în primele trepte de paralelism. De la FP 64 în sus curbele se aplatizează. Dublarea nu mai aduce câștig și introduce cost suplimentar.

Histograma pentru lățime de bandă și pentru numărul de operații pe secundă arată un echilibru specific între citire și scriere. La blocuri de 128 KB și 256 KB scrierea domină constant și depășește 5000 MB pe secundă. Citirea rămâne la valori mai mici între 1000 și 1300 MB pe secundă. Când dimensiunea blocului crește spre 512 KB și 1 MB lățimea de bandă totală scade de la 5500 până la 6400 MB pe secundă la aproximativ 2300 MB pe secundă. IOPS scade la fel de abrupt. Apare o zonă mai puțin eficientă pentru blocuri mari.

Lățimea de bandă medie raportată la FP confirmă aceeași tendință. Curbele cresc rapid de la FP 8 la FP 32 și trec de 5000 MB pe secundă pentru 128 KB și 256 KB. De la FP 64 evoluția se aplatizează. După FP 128 apar oscilații și scăderi. Analiza frontierei dintre lățimea de bandă totală și p99 arată o zonă utilă cu valori între 4000 și 6000 MB pe secundă și p99 sub 5 ms. La FP mari apar dispersii. Unele puncte ating 7000 până la 8000 MB pe secundă însă p99 urcă la 20 până la 50 ms și devin neutilizabile în practică.

Sinteza rezultatelor este coerentă. ZFS cu 12 vdev echivalent RAID0 scalează bine până în jurul FP 32. În acest interval eficiența este ridicată iar latențele rămân sub pragul impus. Dincolo de acest nivel performanța se degradează treptat. Cele mai bune rezultate apar pentru blocuri de 128 KB și 256 KB. Debitul total depășește 6000 MB pe secundă iar IOPS atinge valori mari. p99 rămâne între 2 și 4 ms. ZFS favorizează blocuri mai mici și paralelism mediu. Configurarea trebuie aleasă cu grijă pentru a evita zonele de instabilitate și pentru a păstra latențe acceptabile.

5.3.4.2. ZFS - 6 x 2 vdev (echivalent RAID10) - rezultate experimentale

Lățimea de bandă crește aproape liniar între FP=8 și FP=32 pentru toate valorile recordsize; dincolo de FP=32 nu mai există puncte care respectă $p99 \leq 5$ ms. Practic, „paralelismul util” se oprește la FP≈32: mediile urcă de la ~2-2,5 GB/s (FP=8) la ~4-5 GB/s (FP=32), cu rezultate mai bune la 64k/128k, mai modeste la 1M. Recordsize 256k se apropie de 64k/128k la FP mici, dar rămâne ușor în urmă; 512k e acceptabil doar la FP=32; 1M intră constant pe o pantă inferioară, iar la 2M nu apar deloc puncte sub pragul de latență.

Frontiera debit-latență confirmă limitele: grupul valid ($p99 \approx 3-5$ ms) se află în intervalul ~2-5 GB/s, dominat de 64k/128k (mai rar 256k). Peste 5 ms apar debite mai mari, dar cu p99 de ordinul zecilor de milisecunde, deci neutilizabile în cadrul criteriului de validare. Sinteza MB/s și IOPS, pe direcții, arată un profil în care scrierea domină: vârfuri IOPS >500k pe scriere și <200k pe citire (aliniat profilului de I/O). Pentru recordsize 1M, atât debitul cât și IOPS scad abrupt, confirmând că granularitatea mare este dezavantajoasă.

Recomandări practice: ZFS cu 6 vdev în oglindă scalează bine până la FP≈32 pentru recordsize 64k-128k; acestea sunt opțiunile robuste. Recordsize 256k poate fi folosit ca alternativă când se urmărește un plus modest pe scriere, menținând $p99 \leq 5$ ms. Dimensiunile mari (512k-1M) sunt dezavantajoase, iar 2M este neeligibil din perspectiva latenței. Zonele „de lucru” recomandate: FP 16-32, recordsize 64k-128k (256k opțiune secundară).

5.3.4.3. ZFS - raidz1 - rezultate experimentale

ZFS în configurație raidz1 oferă performanțe competitive doar în anumite combinații de recordsize și paralelism. Frontiera debit-latență arată un interval larg al lățimii de bandă: ~2000 MB/s la blocuri mari (512k-1M) și FP redus, până la >8000 MB/s la blocuri de 64k-128k cu FP în jur de 16×2 sau $32 \times 1/32 \times 2$, menținând p99 în plaja 3-5 ms. Creșterea paralelismului peste 64-128 dispersează rezultatele și împinge p99 spre valori ridicate (până la >50 ms), semn al blocajelor induse de gestionarea parității.

Curbele debit mediu (R+W) vs FP, filtrate la $p99 \leq 5$ ms, cresc constant pentru 64k-128k până la FP=64 (≈6 GB/s pentru 64k și puțin sub 6 GB/s pentru 128k). La 512k-1M,

plafonarea apare devreme, în zona 2-3 GB/s, chiar și la FP=64, confirmând sensibilitatea raidz1 la blocuri mari.

Histograma pe direcții evidențiază dominanța scrierii (profilul de test este orientat spre scriere) și costul parității amortizat prin paralelism moderat: pentru 128k, scrierea depășește 6200 MB/s ($\approx 448k$ IOPS), în timp ce citirea rămâne ~ 1675 MB/s ($\approx 222k$ IOPS); pentru 64k, scrierea este > 5200 MB/s vs ~ 1590 MB/s la citire. La blocuri mari (512k-1M), atât debitul, cât și IOPS scad semnificativ (scriere ≈ 2000 MB/s sau mai puțin; citire < 500 -1000 MB/s), semnalând ineficiența pe acces secvențial de volum mare.

În ansamblu, punctele optime pentru raidz1 se obțin la recordsize 64k-128k și paralelism moderat (≈ 16 -32 fire/cozi), unde se ating 6000-7900+ MB/s cu p99 sub 5 ms și IOPS apropiate de un milion. Dincolo de această zonă, dublarea FP nu mai aduce câștiguri utile și crește latențele. Concluzia operațională este că raidz1 este potrivit pentru sarcini cu blocuri mici și concurență ridicată, dar devine rapid necompetitiv la blocuri mari, unde paritatea și sincronizarea vdev-urilor limitează scalarea.

5.3.5. Sistem OCFS2 cu iSCSI - configurare și rezultate

Capitolul evaluează OCFS2 peste dispozitive bloc expuse prin iSCSI accelerat cu RDMA (iSER), subliniind particularitățile față de scenariile anterioare cu acces local (mdadm, vdev-uri ZFS). Transportul prin RDMA înlocuiește stiva TCP/IP și elimină copiile redundante între spațiul utilizator și nucleu, reducând latența capăt-la-capăt și crescând debitul prin valorificarea capabilităților hardware RDMA. OCFS2 este ales deoarece gestionează nativ accesul concurent distribuit, blocări, jurnalizare distribuită și coerența metadatelor.

Pentru comparabilitate, structurile RAID au fost definite la nivelul dispozitivelor iSCSI (RAID0, RAID10, RAID5), apoi formate cu OCFS2. Dimensiunea segmentelor (64-2048 kB) a fost corelată cu parametrii de aliniere ai sistemului de fișiere, pentru a evita penalizările date de neconcordanța granularităților. Scopul testelor este cuantificarea echilibrului dintre câștigurile de transport oferite de RDMA și costurile inerente unui sistem de fișiere de cluster, pe cele trei tipuri de RAID, cu posibilitatea de comparație directă față de ext4, XFS și ZFS și de delimitare a limitelor practice într-un scenariu multi-nod.

5.3.5.1. RAID0 + OCFS2 - rezultate experimentale

În configurația RAID0 peste OCFS2 prin iSCSI cu RDMA (iSER), performanța atinge valori foarte ridicate, dar cu o variabilitate sensibilă la parametri. Diagrama lățime de bandă vs. p99 arată atât puncte peste 16 GB/s cu p99 < 5 ms, cât și numeroase cazuri în care creșterea debitului împinge latența spre zeci de milisecunde. Dispersia provine din interacțiunea dintre mecanismele de coordonare/jurnalizare ale OCFS2 și transportul iSER: RDMA optimizează transferul, dar nu elimină costurile de sincronizare. Per total, debitele totale sunt mai mari decât în configurațiile locale testate anterior.

Histograma confirmă dominanța de scriere: la chunk 64 KB se ating ~ 13 GB/s și $> 1,6$ M IOPS pe scriere, cu citire de ~ 4 GB/s și sub 1 M IOPS; scrierile depășesc adesea 2 M IOPS, semn că iSER susține fluxuri multe și mici. În funcție de FP apar oscilații pronunțate: cu chunk 1 MB debitul urcă peste 14 GB/s la FP=256, dar cade sub 2 GB/s la FP=1024; în schimb, la 64 kB crește până aproape de 12 GB/s la FP=1024. Concluzia: RAID0 peste OCFS2+iSER poate oferi performanță de vârf sub 5 ms într-o fereastră optimă de FP, însă devine instabil la paralelism foarte mare; câștigurile la dublare sunt neregulate, iar rezultatul final depinde puternic de reglajele de configurare.

5.3.5.2. RAID10 + OCFS2 - rezultate experimentale

În RAID10 peste OCFS2 cu iSCSI/RDMA, profilul de performanță este mai conservator decât la RAID0, dar mai predictibil. Lățimea de bandă împărțită pe direcții confirmă dominanța scrierii (dată de profilul de test), însă RAID10 păstrează debite robuste la scriere în pofida replicării pe oglinzi, iar citirea beneficiază de flexibilitatea accesului din oricare copie. Numărul de operații pe secundă rămâne ridicat și stabil pe intervale largi de paralelism, cu excepția granularității de 1 MB.

Curbele mediei (sub $p99 \leq 5$ ms) cresc constant până la $FP \approx 64-128$, apoi intră în platou sau scad la FP foarte mari (512-1024), penalizarea fiind mai vizibilă decât la RAID0 din cauza scrierilor duplicate. Frontiera lățime de bandă/ $p99$ grupează punctele competitive într-o zonă compactă de $\sim 8-12$ GB/s cu $p99$ sub 2 ms, rareori peste. Practic, cele mai bune rezultate apar cu segmente RAID de 128-256 kB și paralelism moderat (de tip 16×8 , 32×4): debit ridicat, latențe controlate și variabilitate redusă, o opțiune solidă când reziliența și stabilitatea primează.

5.3.5.3. RAID5 + OCFS2 - rezultate experimentale

RAID5 peste OCFS2 cu iSCSI/RDMA (iSER) oferă performanțe vizibil inferioare față de RAID0 și RAID10 pe aceeași arhitectură. Debitele rămân modeste, IOPS se plafonează devreme, iar creșterea paralelismului aduce câștiguri minime. Curbele de câștig la dublarea FP ($G(FP)$) sunt aproape unitare și, la $FP=128-256$, pot deveni subunitare, semn că costurile parității și ale sincronizărilor (OCFS2 + iSER) depășesc beneficiile paralelizării. Frontierele debit-latență grupează punctele RAID5 într-o zonă joasă a debitelor, cu latențe comparabile sau mai mari decât alternativele, fără extensii către valori înalte de lățime de bandă.

Influența dimensiunii segmentului distribuit (chunk) este redusă: curbele pe granularități diferite sunt apropiate, diferențele fiind de ordinul câtorva procente. Spre deosebire de RAID0/RAID10, segmentele mari (1-2 MB) nu aduc avantaje semnificative; cele mai multe puncte eligibile sub $p99 \leq 5$ ms apar la 64 KB și 128 KB, 256-512 KB au apariții sporadice la debite mai mici, 1 MB rămâne limitat, iar 2 MB nu produce puncte conforme cu pragul de latență.

Per ansamblu, configurația se dovedește cea mai restrictivă dintre toate testele: debite scăzute, IOPS limitat, latențe ridicate și scalabilitate aproape inexistentă. Din punct de vedere practic, RAID5 peste OCFS2/iSER este o alegere nepotrivită pentru scenarii care cer debit mare sau latențe mici, rămânând justificabil doar când redundanța primează și performanța brută nu este obiectivul principal. Rezultatele confirmă că penalitățile introduse de calculul parității și de coordonarea distribuită umbresc beneficiile potențiale ale transportului RDMA în acest context.

5.4. Concluzii experimentale

Capitolul prezintă rezultatele experimentale pentru determinarea profilurilor de intrare/ieșire și evaluarea configurațiilor de stocare: de la validarea monitorizării, la extragerea profilurilor și aplicarea lor pe combinații de sisteme de fișiere și niveluri RAID. Pentru comparație, au fost agregate cele mai bune puncte pe fiecare tip de RAID. La RAID0, ierarhia este clară: OCFS2 și XFS oferă cele mai mari debite și IOPS, urmate de ext4, iar ZFS rămâne vizibil sub acestea. Concordanța dintre MB/s și IOPS indică diferențe reale de capacitate sub constrângerea de latență. La RAID10 apar trei tipare: XFS scalează rapid la paralelizare redusă și se stabilizează devreme; ext4 urmează aceeași traiectorie, dar la nivel inferior; OCFS2 pornește sub ext4 și depășește la paralelizare mai mare, însă fără susținere constantă; ZFS acoperă doar o fereastră FP mică-medie și rămâne sub celelalte. XFS și ext4 sunt coerente pe întreg domeniul, în timp ce OCFS2 este mai sensibil la creșterea FP .

Metodologia definită anterior se validează practic: pentru fiecare RAID s-au identificat setări care respectă $p99$ și maximizează debitul raportat la profilul măsurat la hipervizoare. Abordarea, orientată pe date, permite comparații directe și selecția opțiunilor potrivite, ghidând configurarea stocării în funcție de profilul de lucru, de constrângerile de latență și de obiectivul de a maximiza lățimea de bandă.

Capitolul VI - Concluzii generale, contribuții și direcții viitoare

6.1. Concluzii generale

Capitolul concluzionează o cercetare construită unitar, axată pe evaluarea performanței și predictibilității stocării în platforme de virtualizare, susținută de o arhitectură experimentală completă. Punctul de plecare îl reprezintă analiza comparativă a platformelor CMP open-source, realizată pe criterii arhitecturale, de scalare, rețea, stocare, orchestrare, extensibilitate și maturitate, care justifică alegerea OpenNebula pe baze obiective. Pe acest fundament este definită o infrastructură de test integrată, măsurare, control și sincronizare, ce izolează variabilele inutile (inclusiv prin folosirea unui server de stocare în memorie), standardizează ciclul de viață al VM-urilor și corelează evenimentele cu un sistem de semnalizare în timp real, oferind vizibilitate fină asupra comportamentului subsistemului de stocare.

Implementarea software automatizează întreg fluxul: de la inițializarea VM-urilor la colectarea și prelucrarea datelor, cu integrare nativă în orchestrator și monitorizare web în timp real, ceea ce conferă utilitate practică instrumentelor dezvoltate. Metodologia de testare este riguroasă și orientată spre scenarii reale: sincronizare temporală, validare între rulări și dublă perspectivă de măsurare (la hipervizoare și la serverul de stocare) pentru a surprinde transformările de-a lungul traseului I/O. Originalitatea constă în două metode de reconstrucție a profilului I/O la server pornind de la măsurarea la hipervizoare: o aproximare liniară, intuitivă, și o formulare bazată pe minimizarea divergenței Kullback–Leibler, care oferă calibrare matematică precisă.

Rezultatele sunt integrate într-un cadru de evaluare a configurațiilor RAID și a sistemelor de fișiere, cu accent pe stabilitate și predictibilitate, nu pe maxime izolate. Utilizarea percentilei p99 ca filtru de eligibilitate și a diagramelor sintetice delimitează clar zonele de funcționare utile și permite comparații directe între opțiuni. Validarea pe patru configurații distincte evidențiază compromisurile dintre performanță, stabilitate și complexitate și furnizează criterii operaționale de selecție.

Lucrarea este coerentă pe întreg lanțul, analiză conceptuală, proiectare arhitecturală, implementare și validare, fiecare etapă întărind-o pe următoarea. Concluzia majoră este că performanța și stabilitatea nu pot fi apreciate corect fără profiluri reale de utilizare și instrumente capabile să le redea fidel; testele sintetice, izolate, sunt insuficiente. Abordarea propusă, observabilitate extinsă, reconstrucție exactă a profilurilor și testare sistematică, oferă un cadru aplicabil și generalizabil pentru decizii informate privind configurarea stocării. Echilibrul dintre rigoarea teoretică, atenția la implementare și claritatea metodologică conferă demersului atât valoare științifică, cât și relevanță practică și indică direcția în care ar trebui să evolueze evaluarea infrastructurilor virtualizate.

6.2. Contribuții științifice

Cercetarea propusă oferă un ansamblu coerent de contribuții științifice, care combină analiza conceptuală, proiectarea arhitecturală, dezvoltarea software și validarea experimentală într-o metodologie integrată.

1. Analiză comparativă a platformelor CMP open-source - Lucrarea evaluează sistematic OpenStack, CloudStack și OpenNebula, pe criterii de arhitectură, scalare, rețea, stocare, orchestrare, automatizare, telemetrie, ecosistem și maturitate. Alegerea OpenNebula este fundamentată prin echilibrul dintre simplitate, flexibilitate și integrarea extensiilor necesare.

2. Arhitectură experimentală pentru profiluri de intrare/ieșire - Se propune o arhitectură care observă simultan la nivelul hipervizoarelor și la nivelul serverului de stocare, legată de un model standardizat al ciclului de viață al VM. Structura oferă vizibilitate detaliată și bază solidă pentru analiză.

3. Implementare software integrată - Sunt dezvoltate componente pentru orchestrare, monitorizare și analiză: lansare paralelă prin API RPC OpenNebula, server web de monitorizare cu semafor distribuit, scripturi în VM care marchează etapele și module Python

pentru normalizare, grafice și rapoarte. Întregul flux experimental este automatizat și extensibil.

4. Metodologie pentru determinarea și reconstrucția profilurilor - Metoda corelează datele de la hipervizoare cu cele de la server, separă clar sursele și calibrează distribuțiile prin două tehnici complementare: aproximare liniară între distribuții de referință și minimizarea entropiei relative. Sunt prevăzute proceduri stricte de validare prin repetiții și criterii de selecție.

5. Evaluarea configurațiilor pe profiluri reale - Configurarea și compararea soluțiilor de stocare se fac pe baza profilurilor reale, cu p99 al latenței ca filtru principal de eligibilitate. Diagramele corelează paralelismul cu debitul și stabilitatea latenței, facilitând decizii aplicabile și în alte infrastructuri.

6. Determinare experimentală a profilurilor - Profilurile se obțin din măsurători sincronizate în două puncte, cu scenarii rulate de trei ori, verificare vizuală, reexecutare la nevoie și agregare standardizată în CSV. Se folosesc clase canonice de bloc între 512 B și 2 MB, separat pe citire și scriere. Rezultatele evidențiază diferențe între Linux și Windows și efectele formatelor QCOW2 și RAW.

7. Validare pe patru configurații distincte - Metodologia este aplicată pe ext4, XFS, ZFS și OCFS2, cu RAID 0, 10 și 5, local și distribuit. Concluziile arată limitele unor combinații precum RAID5 și confirmă opțiuni echilibrate, de exemplu XFS cu RAID0 sau OCFS2 cu RAID10, în funcție de compromisurile între performanță, stabilitate și complexitate.

6.3. Direcții viitoare de cercetare și dezvoltare

În urma analizelor critice formulate în cadrul concluziilor și a identificării contribuțiilor originale, se pot formula o serie de direcții de dezvoltare ulterioară sau de aprofundarea a unor contribuții, menite să extindă aplicabilitatea rezultatelor obținute și să adreseze limitările observate în cadrul experimentelor.

1. Extinderea profilurilor de intrare/ieșire - Profilurile vor fi rafinate pentru faze suplimentare ale ciclului de viață al mașinilor virtuale, precum funcționare de lungă durată, migrare și snapshot-uri frecvente, inclusiv o fereastră dedicată instanțierii și pornirii. Se propune corelarea profilurilor cu aplicații reale, de tip baze de date, servere web și sarcini big data.

2. Optimizarea RDMA și scenarii hibride - Se urmărește stabilizarea comportamentului la paralelism ridicat prin reglarea parametrilor RDMA, de exemplu dimensiuni de fereastră și agregare. Va fi analizată coexistența NFS peste TCP cu NFS peste RDMA și impactul competiției dintre fluxuri asupra performanței globale.

3. Analize avansate ale latențelor și consistenței - Peste evaluarea p95 și p99, se propune studierea distribuției complete a latențelor, cu metrici suplimentare și rezoluție crescută, pentru a distinge mai clar configurațiile stabile de cele instabile.

4. Teste de durată și corelare cu resursele - Vor fi efectuate teste pe termen lung pentru a observa stabilitatea, fragmentarea, acumularea de metadate și epuizarea cache-ului. Se va integra monitorizarea consumului energetic, pentru recomandări care iau în calcul eficiența energetică.

5. Criteriu compus de evaluare - Se va defini un indicator agregat care combină mai multe mărimi măsurabile, cu ponderi diferite, validat prin compararea cu rezultatele actuale pentru a simplifica deciziile de selecție.

6. Cadru unificat de testare și selecție - Baza comparativă de rezultate va fi formalizată într-un cadru care, plecând de la un profil dat, selectează rapid configurația optimă. Vor fi integrați algoritmi de potrivire automată și generare de recomandări.

7. Automatizarea generării profilurilor - Profilurile vor fi actualizate automat prin colectare continuă din infrastructuri reale, astfel încât cadrele de testare să reflecte în mod curent modelele efective de utilizare.

8. Integrarea cu tehnologii emergente - Metodologia și profilurile vor fi aplicate la NVMe over Fabrics și evaluate pe sisteme de fișiere distribuite moderne, precum Ceph și GlusterFS, pentru a testa transferabilitatea concluziilor din contextul NFS și iSCSI către medii mai complexe.

Referințe bibliografice

- [1] P. Mell și T. Grance, „The NIST Definition of Cloud Computing”, *NIST Special Publication 800-145*, 2011.
- [2] M. Armbrust *et al.*, „A view of cloud computing”, *Commun. ACM*, vol. 53, nr. 4, pp. 50-58, apr. 2010, doi: 10.1145/1721654.1721672.
- [3] M. Mircea și A. Andreescu, „Using Cloud Computing in Higher Education: A Strategy to Improve Agility in the Current Financial Crisis”, *CIBIMA*, pp. 1-15, iun. 2011, doi: 10.5171/2011.875547.
- [4] N. Sultan, „Cloud computing for education: A new dawn?”, *International Journal of Information Management*, vol. 30, nr. 2, pp. 109-116, apr. 2010, doi: 10.1016/j.ijinfomgt.2009.09.004.
- [5] M. T. Baldassarre, D. Caivano, G. Dimauro, E. Gentile, și G. Visaggio, „Cloud Computing for Education: A Systematic Mapping Study”, *IEEE Trans. Educ.*, vol. 61, nr. 3, pp. 234-244, aug. 2018, doi: 10.1109/TE.2018.2796558.
- [6] G. Gercek, N. Saleem, și D. Steel, „Implementing Cloud Based Virtual Computer Network Labs for Online Education: Experiences from a Phased Approach”, *Int. J. Onl. Eng.*, vol. 12, nr. 03, pp. 70-76, mar. 2016, doi: 10.3991/ijoe.v12i03.5564.
- [7] R. Thavi, R. Jhaveri, V. Narwane, B. Gardas, și N. Jafari Navimipour, „Role of cloud computing technology in the education sector”, *JEDT*, vol. 22, nr. 1, pp. 182-213, ian. 2024, doi: 10.1108/JEDT-08-2021-0417.
- [8] M. Sharma, S. Vadalkar, A. Singh, G. Chavan, și K. P. Tsagarakis, „How Does Cloud Computing Adoption Accelerate Education? – Enhancing Academic Results by Integrating Knowledge Management Pre and Post Pandemic”, *International Journal of Human-Computer Interaction*, pp. 1-21, feb. 2025, doi: 10.1080/10447318.2024.2440673.
- [9] C. Udanor *et al.*, „Deploying an Open Source Cloud Computing Infrastructure for Academic Research”, *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 21, nr. 3, pp. 61-74, 2019.
- [10] „Virtual Desktop Infrastructure (VDI) Deployment Using OpenNebula as a Private Cloud”, în *Communications in Computer and Information Science*, Cham: Springer International Publishing, 2020, pp. 440-450. doi: 10.1007/978-3-030-42517-3_33.
- [11] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, și I. Brandic, „Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”, *Future Generation Computer Systems*, vol. 25, nr. 6, pp. 599-616, iun. 2009, doi: 10.1016/j.future.2008.12.001.
- [12] A. Barkat, A. D. Dos Santos, și T. T. Nguyen Ho, „Open Stack and Cloud Stack: Open Source Solutions for Building Public and Private Clouds”, în *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara: IEEE, sep. 2014, pp. 429-436. doi: 10.1109/SYNASC.2014.64.
- [13] Q. Zhang, L. Cheng, și R. Boutaba, „Cloud computing: state-of-the-art and research challenges”, *J Internet Serv Appl*, vol. 1, nr. 1, pp. 7-18, mai 2010, doi: 10.1007/s13174-010-0007-6.
- [14] R. Buyya, C. Vecchiola, și S. T. Selvi, *Mastering cloud computing: foundations and applications programming*. Amsterdam Boston: Morgan Kaufmann, 2013.
- [15] M. Sala-Zárte și L. Colombo-Mendoza, „CLOUD COMPUTING: A REVIEW OF PAAS, IAAS, SAAS SERVICES AND PROVIDERS”, *Lámpsakos*, nr. 7, p. 47, iun. 2012, doi: 10.21501/21454086.844.
- [16] L. E. Sepúlveda-Rodríguez, J. C. Chavarro-Porras, J. A. Sanabria-Ordoñez, H. E. Castro, și J. Matthews, „A survey of virtualization technologies: towards a new taxonomic proposal”, *Ingeniería e Investigación*, vol. 42, nr. 3, 2022.
- [17] R. Younis, M. Iqbal, K. Munir, M. A. Javed, M. Haris, și S. Alahmari, „A Comprehensive Analysis of Cloud Service Models: IaaS, PaaS, and SaaS in the Context of Emerging

- Technologies and Trend”, în *2024 International Conference on Electrical, Communication and Computer Engineering (ICECCE)*, Kuala Lumpur, Malaysia: IEEE, oct. 2024, pp. 1-6. doi: 10.1109/icecce63537.2024.10823401.
- [18] D. Adetunji, „Cloud Computing Abstractions – IaaS, PaaS, FaaS, and SaaS Explained”. Data accesării: 9 mai 2025. [Online]. Disponibil la: <https://www.freecodecamp.org/news/cloud-computing-abstractions-explained/>
- [19] I. Odun-Ayo, M. Ananya, F. Agono, și R. Goddy-Worlu, „Cloud Computing Architecture: A Critical Analysis”, în *2018 18th International Conference on Computational Science and Applications (ICCSA)*, Melbourne, Australia, Australia: IEEE, iul. 2018, pp. 1-7. doi: 10.1109/iccsa.2018.8439638.
- [20] R. Kandpal și V. Kumar, „IaaS Implementation of a Private Cloud using Open Source Technology”, *IJCA*, vol. 70, nr. 20, pp. 30-35, mai 2013, doi: 10.5120/12186-8284.
- [21] O. S. Holovnia și V. P. Oleksiuk, „Selecting cloud computing software for a virtual online laboratory supporting the Operating Systems course”, *CTE Workshop Proc.*, vol. 9, pp. 216-227, mar. 2022, doi: 10.55056/cte.116.
- [22] B. Hou, F. Chen, Z. Ou, R. Wang, și M. Mesnier, „Understanding I/O Performance Behaviors of Cloud Storage from a Client’s Perspective”, *ACM Trans. Storage*, vol. 13, nr. 2, pp. 1-36, mai 2017, doi: 10.1145/3078838.
- [23] S. Shahzadi, M. Iqbal, Z. U. Qayyum, și T. Dagiuklas, „Infrastructure as a service (IaaS): A comparative performance analysis of open-source cloud platforms”, în *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Lund, Sweden: IEEE, iun. 2017, pp. 1-6. doi: 10.1109/CAMAD.2017.8031522.
- [24] A. Vogel, D. Griebler, C. A. F. Maron, C. Schepke, și L. G. Fernandes, „Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack”, în *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Heraklion, Crete, Greece: IEEE, feb. 2016, pp. 672-679. doi: 10.1109/PDP.2016.75.
- [25] C. Chilipirea, G. Laurentiu, M. Popescu, S. Radoveneanu, V. Cernov, și C. Dobre, „A comparison of private cloud systems”, în *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, IEEE, 2016, pp. 139-143.
- [26] S. Ismaeel, A. Miri, D. Chourishi, și S. M. Reza Dibaj, „Open Source Cloud Management Platforms: A Review”, în *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, New York, NY: IEEE, nov. 2015, pp. 470-475. doi: 10.1109/CSCloud.2015.84.
- [27] A. Rahman, R. Mahdavi-Hezaveh, și L. Williams, „A systematic mapping study of infrastructure as code research”, *Information and Software Technology*, vol. 108, pp. 65-77, apr. 2019, doi: 10.1016/j.infsof.2018.12.004.
- [28] M. Łatkowski și R. Nowak, „OpenStack and Google Cloud performance comparison in Infrastructure as a Service model”, 18 octombrie 2022, *arXiv*: arXiv:2210.09691. doi: 10.48550/arXiv.2210.09691.
- [29] D. C. Marinescu, *Cloud computing: theory and practice*, 2nd ed. Boston: Morgan Kaufmann, 2017.
- [30] K. Hashizume, D. G. Rosado, E. Fernández-Medina, și E. B. Fernandez, „An analysis of security issues for cloud computing”, *J Internet Serv Appl*, vol. 4, nr. 1, p. 5, 2013, doi: 10.1186/1869-0238-4-5.
- [31] European Union Agency for Cybersecurity., *ENISA threat landscape 2021: April 2020 to mid July 2021*. LU: Publications Office, 2021. Data accesării: 5 iulie 2025. [Online]. Disponibil la: <https://data.europa.eu/doi/10.2824/324797>
- [32] S. Arora, S. R. Thota, și S. Gupta, „Artificial Intelligence-Driven Big Data Analytics for Business Intelligence in SaaS Products”, în *2024 First International Conference on Pioneering Developments in Computer Science & Digital Technologies (IC2SDT)*, IEEE, 2024, pp. 164-169.

- [33] B. B. Sezer, S. Topal, și U. Nuriyev, „TPPSUPPLY: A traceable and privacy-preserving blockchain system architecture for the supply chain”, *Journal of Information Security and Applications*, vol. 66, p. 103116, 2022.
- [34] P. Mishra și A. K. Somani, „Host managed contention avoidance storage solutions for Big Data”, *J Big Data*, vol. 4, nr. 1, dec. 2017, doi: 10.1186/s40537-017-0080-9.
- [35] S. Ghemawat, H. Gobioff, și S.-T. Leung, „The Google file system”, în *Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing NY USA: ACM, oct. 2003, pp. 29-43. doi: 10.1145/945445.945450.
- [36] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, și A. Rowstron, „Migrating server storage to SSDs: analysis of tradeoffs”, în *Proceedings of the 4th ACM European conference on Computer systems*, Nuremberg Germany: ACM, apr. 2009, pp. 145-158. doi: 10.1145/1519065.1519081.
- [37] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, și D. K. Panda, „Beyond block I/O: Rethinking traditional storage primitives”, în *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, San Antonio, TX, USA: IEEE, feb. 2011, pp. 301-311. doi: 10.1109/hpca.2011.5749738.
- [38] M. Bjørling, J. Axboe, D. Nellans, și P. Bonnet, „Linux block IO: introducing multi-queue SSD access on multi-core systems”, în *Proceedings of the 6th International Systems and Storage Conference on - SYSTOR '13*, Haifa, Israel: ACM Press, 2013, p. 1. doi: 10.1145/2485732.2485740.
- [39] *An In-Depth Overview of NVMe and NVMe-oF - Types of NVMe queues*. Dell Technologies. Data accesării: 9 mai 2025. [Online]. Disponibil la: <https://infohub.delltechnologies.com/en-us//an-in-depth-overview-of-nvme-and-nvme-of/types-of-nvme-queues/>
- [40] S. Koh, J. Jang, C. Lee, M. Kwon, J. Zhang, și M. Jung, „Faster than Flash: An In-Depth Study of System Challenges for Emerging Ultra-Low Latency SSDs”, 15 decembrie 2019, *arXiv*: arXiv:1912.06998. doi: 10.48550/arXiv.1912.06998.
- [41] D. A. Patterson, G. Gibson, și R. H. Katz, „A Case for Redundant Arrays of Inexpensive Disks (RAID)”, *ACM*, 1988.
- [42] A. Silberschatz, P. B. Galvin, și G. Gagne, *Operating system concepts*, 9th ed. Hoboken, N.J Chichester: Wiley, 2013.
- [43] E. Hermanoy, Marcel, și E. Kristianto, „Performance analysis of RAID storage server with software-based accelerator PernixData FVP”, în *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, Bandung - Bali, Indonesia: IEEE, oct. 2016, pp. 1-5. doi: 10.1109/icitsi.2016.7858190.
- [44] A. S. Tanenbaum și H. Bos, *Modern operating systems*, 4. ed. Boston: Prentice Hall, 2015.
- [45] *ext4 Data Structures and Algorithms*. Data accesării: 14 mai 2025. [Online]. Disponibil la: <https://docs.kernel.org/filesystems/ext4/>
- [46] K. D. Fairbanks, „An analysis of Ext4 for digital forensics”, *Digital Investigation*, vol. 9, pp. S118-S130, aug. 2012, doi: 10.1016/j.diin.2012.05.010.
- [47] D. P. Bovet și M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*, 1. ed. Beijing Köln: O'Reilly, 2001.
- [48] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, și L. Vivier, „The new ext4 filesystem: current status and future plans”, 2007, [Online]. Disponibil la: <https://www.kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf>
- [49] R. Nordvik, „Ext4”, în *Mobile Forensics – The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices*, C. Hummert și D. Pawlaszczyk, Ed., Cham: Springer International Publishing, 2022, pp. 41-68. doi: 10.1007/978-3-030-98467-0_2.
- [50] J. Mostek *et al.*, „Porting the SGI XFS File System to Linux”, în *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, San Diego, CA: USENIX Association, iun. 2000. [Online]. Disponibil la: <https://www.usenix.org/conference/2000-usenix-annual-technical-conference/porting-sgi-xfs-file-system-linux>

- [51] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, și G. Peck, „Scalability in the XFS file system”, în *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, în ATEC '96. USA: USENIX Association, 1996, p. 1.
- [52] C. Hellwig, „XFS: the big storage file system for Linux”, Usenix, ;login:, 2009.
- [53] „What Users Should Know About Full Disk Encryption Based on LUKS”, în *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2015, pp. 225-237. doi: 10.1007/978-3-319-26823-1_16.
- [54] S. Whitehouse, „The GFS2 filesystem”, în *Proceedings of the Linux Symposium*, Citeseer, 2007, pp. 253-259.
- [55] M. Fasheh, „OCFS2: The Oracle clustered file system, version 2”, în *Proceedings of the 2006 Linux Symposium*, 2006, pp. 289-302.
- [56] F. Schmuck și R. Haskin, „{GPFS}: A {Shared-Disk} file system for large computing clusters”, în *Conference on file and storage technologies (FAST 02)*, 2002.
- [57] S. Weil, S. A. Brandt, E. L. Miller, D. D. Long, și C. Maltzahn, „Ceph: A scalable, high-performance distributed file system”, în *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*, 2006, pp. 307-320.
- [58] Ceph. Data accesării: 16 mai 2025. [Online]. Disponibil la: <https://docs.ceph.com/en/reef/>
- [59] OpenZFS Guide. Data accesării: 16 mai 2025. [Online]. Disponibil la: <https://openzfs.readthedocs.io/en/latest>
- [60] J. Bonwick, M. Ahrens, V. Henson, M. Maybee, și M. Shellenbaum, „The zettabyte file system”, în *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, 2003, p. 1.
- [61] V. Dakic, M. Kovac, și I. Videc, „High-Performance Computing Storage Performance and Design Patterns—Btrfs and ZFS Performance for Different Use Cases”, *Computers*, vol. 13, nr. 6, p. 139, iun. 2024, doi: 10.3390/computers13060139.
- [62] *The Z File System (ZFS)*. Data accesării: 12 mai 2025. [Online]. Disponibil la: <https://docs.freebsd.org/en/books/handbook/zfs/>
- [63] S. Mushran, „OCFS2”, 2008, [Online]. Disponibil la: https://www.academia.edu/11510892/OCFS2_A_CLUSTER_FILE_SYSTEM_FOR_LINUX_USERS_GUIDE_FOR_RELEASE_1_6_Sunil_Mushran
- [64] „Managing the Oracle Cluster File System Version 2”. Oracle, 2025. [Online]. Disponibil la: <https://docs.oracle.com/en/operating-systems/oracle-linux/10/ocfs2/OL10-OCFS2.pdf>
- [65] R. Sandberg, „The Sun Network Filesystem: Design, Implementation and Experience”, 2001. [Online]. Disponibil la: <https://api.semanticscholar.org/CorpusID:60222151>
- [66] J. Parisi și B. R. Choudhury, „NFSv4: Enhancements and Best Practices Guide”, NetApp, feb. 2016. [Online]. Disponibil la: <https://www.netapp.com/media/16398-tr-3580.pdf>
- [67] D. Hildebrand, P. Honeyman, și W. A. Adamson, „pNFS and Linux: Working Towards a Heterogeneous Future”, în *Proceedings the 8th LCI International Conference on High-Performance Clustered Computing*, Lake Tahoe, CA, mai 2007.
- [68] B. Callaghan, T. Lingutla-Raj, A. Chiu, P. Staubach, și O. Asad, „NFS over RDMA”, în *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, Karlsruhe Germany: ACM, aug. 2003, pp. 196-208. doi: 10.1145/944747.944753.
- [69] R. K. Khattar, M. S. Murphy, G. J. Tarella, și K. E. Nystrom, „Introduction to Storage Area Network, SAN”, *IBM Redbooks*, 1999.
- [70] J. S. Glider, C. F. Fuente, și W. J. Scales, „The software architecture of a SAN storage control system”, *IBM Syst. J.*, vol. 42, nr. 2, pp. 232-249, 2003, doi: 10.1147/sj.422.0232.
- [71] A. Khan, „Fibre Channel Functional Overview”, *SANOG 8*, 2006, [Online]. Disponibil la: <https://www.sanog.org/resources/sanog8/sanog8-san-functional-overview-asimkhan.pdf>
- [72] M. Chadalapaka, J. Satran, K. Meth, și D. L. Black, „Internet Small Computer System Interface (iSCSI) Protocol (Consolidated)”, nr. 7143. în Request for Comments. RFC Editor, aprilie 2014. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc7143>

- [73] M. Oguchi, R. Higa, K. Matsubara, T. Okamawari, și S. Yamaguchi, „Performance Improvement of iSCSI Remote Storage Access through Optimization of Multiple Layers”, *JSW*, vol. 8, nr. 3, mar. 2013, doi: 10.4304/jsw.8.3.538-546.
- [74] S. Racherla, S. Erdenberger, H. Rajagopal, și K. Ruth, „Storage and Network Convergence Using FCoE and iSCSI”, *IBM Redbooks*, [Online]. Disponibil la: <https://www.redbooks.ibm.com/redbooks/pdfs/sg247986.pdf>
- [75] S. Aiken, D. Grunwald, A. R. Pleszkun, și J. Willeke, „A performance analysis of the iSCSI protocol”, în *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings.*, San Diego, CA, USA: IEEE Comput. Soc, pp. 123-134. doi: 10.1109/mass.2003.1194849.
- [76] R. Mittal *et al.*, „Revisiting network support for RDMA”, în *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, Budapest Hungary: ACM, aug. 2018, pp. 313-326. doi: 10.1145/3230543.3230557.
- [77] T. Hoefler *et al.*, „Datacenter ethernet and rdma: Issues at hyperscale”, *arXiv preprint arXiv:2302.03337*, 2023.
- [78] „RDMA Technology White Paper”, H3C. [Online]. Disponibil la: https://www.h3c.com/en/Support/Resource_Center/EN/Home/Switches/00-Public/Trending/Technology_White_Papers/RDMA_Technology_White_Paper-6W100/
- [79] L. Chen, „Intelligent and Lossless Network (HPC Scenario)”, *Huawei Technologies Co*, 2021.
- [80] X. Kong *et al.*, „Understanding {RDMA} microarchitecture resources for performance isolation”, în *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 31-48.
- [81] Z. Wang, X. Wang, Z. Qian, B. Ye, și S. Lu, „Rdmavisor: Toward deploying scalable and simple RDMA as a service in datacenters”, *arXiv preprint arXiv:1802.01870*, 2018.
- [82] P. MacArthur, Q. Liu, R. D. Russell, F. Mizero, M. Veeraraghavan, și J. M. Dennis, „An Integrated Tutorial on InfiniBand, Verbs, and MPI”, *IEEE Commun. Surv. Tutorials*, vol. 19, nr. 4, pp. 2894-2926, 2017, doi: 10.1109/comst.2017.2746083.
- [83] „RoCE vs. iWARP A Great Storage Debate”. SNIA ESF, august 2018. [Online]. Disponibil la: <https://www.snia.org/sites/default/files/ESF/RoCE-vs.-iWARP-Final.pdf>
- [84] W. Nouredine, „NFS over 40Gbps iWARP RDMA”, SNIA, 2014. [Online]. Disponibil la: https://www.snia.org/sites/default/files/WaelNouredine__NFSoverRDMA_Final-Rev.pdf
- [85] R. J. Recio, P. R. Culley, D. Garcia, B. Metzler, și J. Hilland, „A Remote Direct Memory Access Protocol Specification”, nr. 5040. în Request for Comments. RFC Editor, octombrie 2007. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc5040>
- [86] P. R. Culley, S. Bailey, J. Carrier, U. Elzur, și R. J. Recio, „Marker PDU Aligned Framing for TCP Specification”, nr. 5044. în Request for Comments. RFC Editor, octombrie 2007. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc5044>
- [87] P. R. Culley, H. Shah, J. Pinkerton, și R. J. Recio, „Direct Data Placement over Reliable Transports”, nr. 5041. în Request for Comments. RFC Editor, octombrie 2007. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc5041>
- [88] M. Hagen și R. Zarick, „Performance evaluation of dCB's priority-Based flow control”, în *2011 IEEE 10th International Symposium on Network Computing and Applications*, IEEE, 2011, pp. 328-333.
- [89] Y. Cai, Y. Yan, Z. Zhang, și Y. Yang, „Survey on converged data center networks with DCB and FCoE: standards and protocols”, *IEEE Network*, vol. 27, nr. 4, pp. 27-32, 2013.
- [90] C. Guo *et al.*, „RDMA over commodity ethernet at scale”, în *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202-215.
- [91] S. Vegesna, *IP quality of service*. Cisco press, 2001.
- [92] J. Geng, J. Yan, și Y. Zhang, „P4QCN: Congestion Control Using P4-Capable Device in Data Center Networks”, *Electronics*, vol. 8, nr. 3, 2019, doi: 10.3390/electronics8030280.
- [93] *IEEE 802.1 Data Center Bridging*. [Online]. Disponibil la: <https://www.ieee802.org/1/pages/dcbridges.html>

- [94] S. Floyd, D. K. K. Ramakrishnan, și D. L. Black, „The Addition of Explicit Congestion Notification (ECN) to IP”, nr. 3168. în Request for Comments. RFC Editor, septembrie 2001. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc3168>
- [95] Microsoft, *SMB Direct*. (2025). [Online]. Disponibil la: <https://learn.microsoft.com/en-us/windows-server/storage/file-server/smb-direct>
- [96] Y. Jia, E. Anger, și F. Chen, „When NVMe over Fabrics Meets Arm: Performance and Implications”, în *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, 2019, pp. 134-140. doi: 10.1109/MSST.2019.000-9.
- [97] C. Lever, W. A. Simpson, și T. Talpey, „Remote Direct Memory Access Transport for Remote Procedure Call Version 1”, nr. 8166. în Request for Comments. RFC Editor, iunie 2017. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc8166>
- [98] R. Noronha, L. Chai, T. Talpey, și D. K. Panda, „Designing NFS with RDMA for Security, Performance and Scalability”, în *2007 International Conference on Parallel Processing (ICPP 2007)*, Xian, China: IEEE, sep. 2007, pp. 49-49. doi: 10.1109/icpp.2007.34.
- [99] M. Ko și A. Nezhinsky, „Internet Small Computer System Interface (iSCSI) Extensions for the Remote Direct Memory Access (RDMA) Specification”, nr. 7145. în Request for Comments. RFC Editor, aprilie 2014. [Online]. Disponibil la: <https://www.rfc-editor.org/info/rfc7145>
- [100] E. Burns și R. Russell, „Implementation and Evaluation of iSCSI over RDMA”, în *2008 Fifth IEEE International Workshop on Storage Network Architecture and Parallel I/Os*, IEEE, 2008, pp. 3-10.
- [101] M. Chadalapaka, H. Shah, U. Elzur, P. Thaler, și M. Ko, „A study of iSCSI extensions for RDMA (iSER)”, în *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, 2003, pp. 209-219.
- [102] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, și R. McQuaid, „Developing cyber-resilient systems : a systems security engineering approach”, National Institute of Standards and Technology, Gaithersburg, MD, dec. 2021. doi: 10.6028/nist.sp.800-160v2r1.
- [103] D. Egan, A. Kurmi, P. Rich, M. Roza, și M. Schrock, „Key Management in Cloud Services”. Cloud Security Alliance, 2020. [Online]. Disponibil la: <https://cloudsecurityalliance.org/artifacts/key-management-when-using-cloud-services>
- [104] T. L. Moore, S. S. Conlon, și A. U. Hewarathna, „Encryption Methods and Key Management Services for Secure Cloud Computing: A Review”.
- [105] R. Snouffer și A. Oldehoeft, „A COMPARISON OF THE SECURITY REQUIREMENTS”, *NIST Special Publication 800-29*, iun. 2001.
- [106] Y. Zhang, A. Rajimwale, A. C. Arpaci-Dusseau, și R. H. Arpaci-Dusseau, „End-to-end Data Integrity for File Systems: A ZFS Case Study.”, în *FAST*, 2010, pp. 29-42.
- [107] S. Narayan, J. A. Chandy, S. Lang, P. Carns, și R. Ross, „Uncovering errors: the cost of detecting silent data corruption”, în *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, Portland Oregon: ACM, nov. 2009, pp. 37-41. doi: 10.1145/1713072.1713083.
- [108] E. Bocchi, A. Lekshmanan, R. Valverde, și Z. Goggin, „Enabling Storage Business Continuity and Disaster Recovery with Ceph distributed storage”, în *EPJ Web of Conferences*, EDP Sciences, 2024, p. 01021.
- [109] P. Loscocco și S. D. Smalley, „Meeting critical security objectives with security-enhanced linux”, în *Proceedings of the 2001 Ottawa Linux symposium*, 2001, pp. 115-134.
- [110] D. Hitaj, G. Pagnotta, F. De Gaspari, L. De Carli, și L. V. Mancini, „Minerva: A file-based ransomware detector”, *arXiv preprint arXiv:2301.11050*, 2023.
- [111] C. Constantinescu și S. Seshadri, „Sentinel: Ransomware detection in file storage”, în *Proceedings of the 14th ACM International Conference on Systems and Storage*, 2021, pp. 1-1.
- [112] E. Berrueta, D. Morato, E. Magaña, și M. Izal, „Crypto-ransomware detection using machine learning models in file-sharing network scenarios with encrypted traffic”, *Expert Systems with Applications*, vol. 209, p. 118299, 2022.

- [113] D. Min și Y. Kim, „Isolating namespace and performance in key-value SSDs for multi-tenant environments”, în *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*, 2021, pp. 8-13.
- [114] *Using Container Storage Interface*. Data accesării: 3 mai 2025. [Online]. Disponibil la: https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/storage/using-container-storage-interface-csi
- [115] *Kubernetes security mechanism*. Data accesării: 17 mai 2025. [Online]. Disponibil la: <https://kubernetes.io/docs/concepts/security>
- [116] S. Ma, T. Ma, K. Chen, și Y. Wu, „A survey of storage systems in the RDMA era”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, nr. 12, pp. 4395-4409, 2022.
- [117] *OpenStack*. Data accesării: 17 mai 2025. [Online]. Disponibil la: <https://www.openstack.org/software/>
- [118] *Apache CloudStack*. Data accesării: 4 mai 2025. [Online]. Disponibil la: <https://cloudstack.apache.org>
- [119] *Apache CloudStack Documentation*. Data accesării: 6 mai 2025. [Online]. Disponibil la: <https://docs.cloudstack.apache.org/en>
- [120] *OpenNebula*. Data accesării: 4 mai 2025. [Online]. Disponibil la: <https://opennebula.io/discover/>
- [121] F. F. Wong și C. X. Tan, „ANALYSIS OF CLOUD COMPUTING IN OPEN SOURCE PLATFORM”, în *Proceedings of the 1st International Conference on Cloud Computing and Services Science*, Noordwijkerhout, Netherlands: SciTePress - Science and Technology Publications, 2011, pp. 158-165. doi: 10.5220/0003382701580165.
- [122] T. Lynn, G. Hunt, D. Corcoran, J. Morrison, și P. Healy, „A Comparative Study of Current Open-source Infrastructure as a Service Frameworks”, în *Proceedings of the 5th International Conference on Cloud Computing and Services Science*, Lisbon, Portugal: SCITEPRESS - Science and Technology Publications, 2015, pp. 95-104. doi: 10.5220/0005423300950104.
- [123] P. Sempolinski și D. Thain, „A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus”, în *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, Indianapolis, IN, USA: IEEE, nov. 2010, pp. 417-426. doi: 10.1109/cloudcom.2010.42.
- [124] J. P. Mullerikkal și Y. Sastri, „A Comparative Study of OpenStack and CloudStack”, în *2015 Fifth International Conference on Advances in Computing and Communications (ICACC)*, Kochi, India: IEEE, sep. 2015, pp. 81-84. doi: 10.1109/ICACC.2015.110.
- [125] V. N. Van, L. M. Chi, N. Q. Long, G. N. Nguyen, și D.-N. Le, „A Performance Analysis of OpenStack Open-Source Solution for IaaS Cloud Computing”, în *Proceedings of the Second International Conference on Computer and Communication Technologies*, vol. 380, S. C. Satapathy, K. S. Raju, J. K. Mandal, și V. Bhateja, Ed., în *Advances in Intelligent Systems and Computing*, vol. 380, New Delhi: Springer India, 2016, pp. 141-150. doi: 10.1007/978-81-322-2523-2_13.
- [126] A. Paradowski, L. Liu, și B. Yuan, „Benchmarking the Performance of OpenStack and CloudStack”, în *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, Reno, NV, USA: IEEE, iun. 2014, pp. 405-412. doi: 10.1109/isorc.2014.12.
- [127] D. Freet, R. Agrawal, J. J. Walker, și Y. Badr, „Open source cloud management platforms and hypervisor technologies: A review and comparison”, în *SoutheastCon 2016*, Norfolk, VA, USA: IEEE, mar. 2016, pp. 1-8. doi: 10.1109/secon.2016.7506698.
- [128] M. M. A. AL-Mukhtar și A. A. Ali Mardan, „Performance Evaluation of the CloudStack Private Cloud”, *IJ-CLOSER*, vol. 2, nr. 6, feb. 2014, doi: 10.11591/closer.v2i6.5661.
- [129] A. B. M. Moniruzzaman, K. W. Nafi, și S. A. Hossain, „An experimental study of load balancing of OpenNebula open-source cloud computing platform”, în *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, Dhaka, Bangladesh: IEEE, mai 2014, pp. 1-6. doi: 10.1109/ICIEV.2014.6850733.

- [130] M. Jones *et al.*, „Scalability of VM provisioning systems”, în *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA: IEEE, sep. 2016, pp. 1-5. doi: 10.1109/HPEC.2016.7761629.
- [131] *OpenNebula 7.0 Documentation*. Data accesării: 4 iulie 2025. [Online]. Disponibil la: <https://docs.opennebula.io/7.0/>
- [132] Y. Wang, J. Xing, J. Xiong, și D. Meng, „A Load-Aware Data Placement Policy on Cluster File System”, în *Network and Parallel Computing*, E. Altman și W. Shi, Ed., în *Lecture Notes in Computer Science*, vol. 6985. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 17-31. doi: 10.1007/978-3-642-24403-2_2.
- [133] F. Z. Boito, R. Kassick, P. O. A. Navaux, și Y. Denneulin, „Towards fast profiling of storage devices regarding access sequentiality”, în *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Salamanca Spain: ACM, apr. 2015, pp. 2015-2020. doi: 10.1145/2695664.2695701.
- [134] S. Kavalanekar, B. Worthington, Qi Zhang, și V. Sharda, „Characterization of storage workload traces from production Windows Servers”, în *2008 IEEE International Symposium on Workload Characterization*, Seattle, WA, USA: IEEE, oct. 2008, pp. 119-128. doi: 10.1109/IISWC.2008.4636097.
- [135] H. Lee și G. Fox, „Big Data Benchmarks of High-Performance Storage Systems on Commercial Bare Metal Clouds”, în *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy: IEEE, iul. 2019, pp. 1-8. doi: 10.1109/CLOUD.2019.00014.
- [136] J. Axboe, *Flexible I/O tester synthetic benchmark - FIO*. [Online]. Disponibil la: <https://github.com/axboe/fio>
- [137] Q. Noorshams, A. Busch, S. Kounev, și R. Reussner, „The Storage Performance Analyzer: Measuring, Monitoring, and Modeling of I/O Performance in Virtualized Environments (Invited Demonstration Paper)”, în *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, Austin Texas USA: ACM, ian. 2015, pp. 107-108. doi: 10.1145/2668930.2693845.
- [138] *Filebench*. Data accesării: 16 mai 2025. [Online]. Disponibil la: <https://github.com/filebench/filebench>
- [139] J. Li, Q. Wang, P. P. C. Lee, și C. Shi, „An In-Depth Comparative Analysis of Cloud Block Storage Workloads: Findings and Implications”, 19 noiembrie 2022, *arXiv*: arXiv:2203.10766. doi: 10.48550/arXiv.2203.10766.
- [140] D. Ellard și M. Seltzer, „New NFS tracing tools and techniques for system analysis”, în *Proceedings of the 17th large installation systems administration conference*, USENIX Association, 2003.
- [141] E. Anderson, „Capture, Conversion, and Analysis of an Intense NFS Workload.”, în *FAST*, 2009, pp. 139-152.
- [142] M. Chen *et al.*, „Linux NFSv4.1 Performance Under a Microscope”.
- [143] A. Kangarlou, S. Shete, și J. D. Strunk, „Chronicle: Capture and Analysis of {NFS} Workloads at Line Rate”, în *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 345-358.
- [144] D. Ellard și J. Ledlie, „Passive {NFS} tracing of email and research workloads”, în *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, 2003.
- [145] *libvirt*. Data accesării: 25 mai 2025. [Online]. Disponibil la: <https://libvirt.org/docs.html>
- [146] *collectl*. Data accesării: 2 mai 2025. [Online]. Disponibil la: <https://collectl.sourceforge.net/>
- [147] P. Snyder, „tmpfs: A Virtual Memory File System”, în *Proceedings of the autumn 1990 EUUG Conference*, 1990.
- [148] E. Jacobson, P. Schliep, și C. Holmes, „Reducing Memory Footprint for In-Memory Root Filesystem (TMPFS)”, 2019, [Online]. Disponibil la: https://www.tdcommons.org/dpubs_series/1864/
- [149] Hao Li, Yongping Xiong, și Jian Ma, „Overtmpfs: A virtual memory file system based on tmpfs”, în *Proceedings of 2011 International Conference on Computer Science and*

- Network Technology*, Harbin, China: IEEE, dec. 2011, pp. 2712-2715. doi: 10.1109/ICCSNT.2011.6182526.
- [150] L. Kosta, H. Hunter, G. George, A. Strelzoff, și S. J. Matthews, „Measuring I/O Performance of Lustre and the Temporary File System for Tradespace Applications on HPC Systems”, în *Proceedings of the SouthEast Conference*, Kennesaw GA USA: ACM, apr. 2017, pp. 187-190. doi: 10.1145/3077286.3077326.
- [151] R. Love, *Linux kernel development: a thorough guide to the design and implementation of the Linux kernel*, 3. ed., 3. printing. în Developer's library. Upper Saddle River, NJ: Addison-Wesley, 2011.
- [152] J. Corbet, A. Rubini, și G. Kroah-Hartman, *Linux Device Drivers*, 3rd ed. Sebastopol: O'Reilly Media, Inc, 2009.
- [153] J. Axboe, „Linux block IO—present and future”, prezentat la Ottawa Linux Symp, 2004, pp. 51-61.
- [154] *ramfs, rootfs, initramfs*. Data accesării: 24 mai 2025. [Online]. Disponibil la: <https://docs.kernel.org/filesystems/ramfs-rootfs-initramfs.html>
- [155] *zram*. Data accesării: 18 mai 2025. [Online]. Disponibil la: <https://docs.kernel.org/admin-guide/blockdev/zram.html>
- [156] F. Douglist, „The Compression Cache: Using On-line Compression to Extend Physical Memory.”, *Usenix Winter*, vol. 92, pp. 1-16, 1993.
- [157] Se-Jun Kwon, Sang-Hoon Kim, Hyeong-Jun Kim, și Jin-Soo Kim, „LZ4m: A fast compression algorithm for in-memory data”, în *2017 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA: IEEE, 2017, pp. 420-423. doi: 10.1109/ICCE.2017.7889380.
- [158] D. Ying și Y. Nian-min, „The analytical methods based on the distributed storage system I/O performance model”, în *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, Zhangjiajie, China: IEEE, mai 2012, pp. 509-512. doi: 10.1109/CSAE.2012.6272648.
- [159] G. Haas și V. Leis, „What Modern NVMe Storage Can Do, and How to Exploit it: High-Performance I/O for High-Performance Storage Engines”, *Proc. VLDB Endow.*, vol. 16, nr. 9, pp. 2090-2102, mai 2023, doi: 10.14778/3598581.3598584.
- [160] E. Borba, E. Tavares, și P. Maciel, „A modeling approach for estimating performance and energy consumption of storage systems”, *Journal of Computer and System Sciences*, vol. 128, pp. 86-106, sep. 2022, doi: 10.1016/j.jcss.2022.04.001.
- [161] S. Zheng, M. Hoseinzadeh, S. Swanson, și L. Huang, „TPFS: A High-Performance Tiered File System for Persistent Memories and Disks”, *ACM Trans. Storage*, vol. 19, nr. 2, pp. 1-28, mai 2023, doi: 10.1145/3580280.
- [162] A. Davis *et al.*, „In Praise of Memory Systems: Cache, DRAM, Disk”, în *Memory Systems*, Elsevier, 2008, pp. i-ii. doi: 10.1016/B978-0-12-379751-3.50036-9.
- [163] D. K. Sung, Y. Son, H. Eom, și S. Kim, „Improving I/O Performance via Address Remapping in NVMe Interface”, *IEEE Access*, vol. 10, pp. 119722-119733, 2022, doi: 10.1109/ACCESS.2022.3221733.
- [164] Q. Xu *et al.*, „Performance analysis of NVMe SSDs and their implication on real world databases”, în *Proceedings of the 8th ACM International Systems and Storage Conference*, Haifa Israel: ACM, mai 2015, pp. 1-11. doi: 10.1145/2757667.2757684.
- [165] H. Liu *et al.*, „A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions”, 9 octombrie 2020, *arXiv*: arXiv:2010.04406. doi: 10.48550/arXiv.2010.04406.
- [166] B. Tabatabai, M. Mansi, și M. M. Swift, „FBMM: Using the VFS for Extensibility in Kernel Memory Management”, în *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*, Providence RI USA: ACM, iun. 2023, pp. 181-187. doi: 10.1145/3593856.3595908.
- [167] *Overview of the Linux Virtual File System*. Data accesării: 27 mai 2025. [Online]. Disponibil la: <https://docs.kernel.org/filesystems/vfs.html>

- [168] F. Werner, H. Cristopher, și W. Richard, „The Linux Storage Diagram”. [Online]. Disponibil la: https://www.thomas-krenn.com/en/wiki/Linux_Storage_Stack_Diagram
- [169] W. de Bruijn și H. Bos, „PipesFS: fast Linux I/O in the unix tradition”, *SIGOPS Oper. Syst. Rev.*, vol. 42, nr. 5, pp. 55-63, iul. 2008, doi: 10.1145/1400097.1400104.
- [170] R. Moreno-Vozmediano, Montero, și I. M. Llorente, „IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures”, *Computer*, vol. 45, nr. 12, pp. 65-72, dec. 2012, doi: 10.1109/MC.2012.76.
- [171] X. Wen, G. Gu, Q. Li, Y. Gao, și X. Zhang, „Comparison of open-source cloud management platforms: OpenStack and OpenNebula”, în *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, Chongqing, Sichuan, China: IEEE, mai 2012, pp. 2457-2461. doi: 10.1109/FSKD.2012.6234218.
- [172] M. Troutman, „Comparing CloudStack, OpenNebula & OpenStack”. Data accesării: 4 aprilie 2025. [Online]. Disponibil la: <https://linbit.com/blog/comparing-cloudstack-opennebula-openstack/>
- [173] *OpenStack Ceilometer*. Data accesării: 15 mai 2025. [Online]. Disponibil la: docs.openstack.org/ceilometer/latest
- [174] *OpenStack Mistral*. Data accesării: 15 mai 2025. [Online]. Disponibil la: <https://docs.openstack.org/mistral/latest/>
- [175] *Apache CloudStack Plugins Guide*. Data accesării: 16 mai 2025. [Online]. Disponibil la: <https://docs.cloudstack.apache.org/en/4.20.0.0/plugins/index.html>
- [176] *CloudStack Java API*. Data accesării: 15 mai 2025. [Online]. Disponibil la: <https://github.com/apache/cloudstack/blob/master/api/src/main/java/org/apache/cloudstack/api/ApiCommandResourceType.java>
- [177] *CloudStack Async Jobs*. Data accesării: 15 mai 2025. [Online]. Disponibil la: <https://cloudstack.apache.org/api/apidocs-4.18/apis/listAsyncJobs.html>
- [178] M. Lee și Y. I. Eom, „Designing Transactional Hypervisor for Improving the Performance of Qcow2-Based Virtual Disks”, *IEEE Access*, vol. 8, pp. 192631-192642, 2020, doi: 10.1109/ACCESS.2020.3033030.
- [179] *OpenNebula - Image Template*. Data accesării: 20 mai 2025. [Online]. Disponibil la: https://docs.opennebula.io/6.10/management_and_operations/references/img_template.html
- [180] *Linux CPUFreq Governors*. Data accesării: 23 mai 2025. [Online]. Disponibil la: <https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt>
- [181] *CPU Performance Scaling*. Data accesării: 24 mai 2025. [Online]. Disponibil la: <https://docs.kernel.org/admin-guide/pm/cpufreq.html>
- [182] J. Manner și G. Wirtz, „Why Many Benchmarks Might Be Compromised”, în *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Oxford, United Kingdom: IEEE, aug. 2021, pp. 100-106. doi: 10.1109/SOSE52839.2021.00016.
- [183] X. Lu, Q. Wu, N. S. V. Rao, și Zongmin Wang, „On Performance-Adaptive flow control for large data transfer in high speed networks”, în *2009 IEEE 28th International Performance Computing and Communications Conference*, Scottsdale, AZ, USA: IEEE, dec. 2009, pp. 49-56. doi: 10.1109/PCCC.2009.5403835.
- [184] *NVIDIA ConnectX-4 Lx Ethernet Adapter Cards User Manual*. Data accesării: 25 mai 2025. [Online]. Disponibil la: <https://docs.nvidia.com/networking/display/cx4lxe>
- [185] G. P. Patil și C. R. Rao, „Weighted Distributions and Size-Biased Sampling with Applications to Wildlife Populations and Human Families”, *Biometrics*, vol. 34, nr. 2, pp. 179-189, 1978.
- [186] C. R. Rao, „Weighted Distributions Arising Out of Methods of Ascertainment: What Population Does a Sample Represent?”, în *A Celebration of Statistics*, A. C. Atkinson și S. E. Fienberg, Ed., New York, NY: Springer New York, 1985, pp. 543-569.
- [187] J.-C. Deville și C.-E. Särndal, „Calibration estimators in survey sampling”, *Journal of the American statistical Association*, vol. 87, nr. 418, pp. 376-382, 1992.
- [188] T. M. Cover și J. A. Thomas, *Elements of Information Theory*, 2nd Edition. în Telecommunications and Signal Processing. Wiley-Interscience, 2006.

- [189]S. P. Boyd și L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [190]W. H. Press, S. A. Teukolsky, W. T. Vetterling, și B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. USA: Cambridge University Press, 2006.
- [191]R. Werner, D. Valev, și D. Danov, „The pearson's correlation-a measure for the linear relationships between time series”, *Fundamental space research*, 2009.

ANEXE

Datorită volumului foarte mare de diagrame care au fost generate și faptului că au fost scrise un număr mare de programe și scripturi, toate acestea ar încărca semnificativ porțiunea de anexe din lucrare.

Toate cele menționate anterior pot fi accesate pentru consultare la adresa:

<https://aistrate.cti.ugal.ro/doctorat>

folosind următoarele coordonate de autentificare:

Username: doctorat

Password: 2025