Universitatea Națională de Științe și Tehnologie POLITEHNICA București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare



# TEZĂ DE DOCTORAT

## Rezumat

# Îmbunătățirea execuției aplicațiilor în cluster, Grid și cloud

**Conducător Științific:**
Prof. Dr. Ing. Nicolae ȚĂPUȘ

**Autor:**
drd. Maria-Elena MIHĂILESCU

București, 2024

National University of Science and Technology POLITEHNICA Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department



# PHD THESIS

## Summary

# Improvements to application execution in cluster, Grid and cloud environments

**Scientific Adviser:**
Prof. Dr. Ing. Nicolae ȚĂPUȘ

**Author:**
drd. Maria-Elena MIHĂILESCU

Bucharest, 2024

# Abstract

Pe măsură ce domeniul științific evoluează, apar și noi subiecte de cercetare. Multe dintre acestea au nevoie de multă putere computațională pentru a fi rezolvate. Totodată, există o tendință a migrării serviciilor din infrastructura locală către soluții de tip cloud. Astfel, apare nevoia de arhitecturi complexe care să furnizeze resurse (calcul, stocare, rețea) pe baza nevoilor utilizatorilor. Aceste resurse sunt, de obicei, grupate în arhitecturi de tip cluster sau Grid. Abstractizând accesul la resurse la aceste două tipuri de arhitecturi, se obțin arhitecturile de tip cloud.

Cu toate acestea, arhitecturile complexe au nevoie de mentenanță de specialitate pentru a avea performanțe înalte: de la alegerea componentelor și modul lor de interacțiune până la asigurarea securității infrastructurii și satisfacerea nevoilor utilizatorilor. Această teză propune soluții menite să îmbunătățească funcționalitatea în medii cluster, Grid și cloud. Soluțiile propuse sunt folosite în arhitecuturi de tip cluster și cloud, precum cea existentă în Universitatea Națională de Știință și Tehnologie Politehnica București (UNSTPB), dar și în structuri Grid, precum Gridul ALICE de la Organizația Europeană pentru Cercetări Nucleare (CERN).

ALICE este unul din cele patru experimente ale proiectului Large Hadron Collider (LHC) de la CERN. Datele rezultate în urma experimentelor fizice sunt procesate folosind resurse complexe (până la 200.000 core-uri). Astfel, procesările de date existente în Grid trebuie planificate în mod eficient. Fiecare procesare are atribuită un timp maxim de rulare (TTL = Time To Live), iar intervalele de execuție din Grid au, de asemenea, o durată maximă în care sunt deschise (de obicei, 24h). Din cauza eterogenității mari a Gridului, TTL-ul este configurat astfel încât cererile să poată să își termine execuția și pe cele mai lente sisteme. Însă, această metodă nu este optimă deoarece stațiile de lucru cu performanțe mai bune nu vor putea împărți eficient intervalele de execuție din cauza TTL-ului mult prea mare. Ajustând TTL-ul dinamic la o valoare mai bună pentru anumite sisteme, se poate îmbunătății procesul de planificare al cererilor în Grid. Această teză analizează folosirea resurselor în cadrul simulărilor de tip Monte Carlo din Grid-ul ALICE și arată factorii care influențează execuția. De asemenea, propune un algoritm de estimare a TTL-ului în funcție de factorii determinați și arată rezultatele obținute în cadrul Grid-ului ALICE.

În ultimii ani, se observă o migrare spre folosirea (totală sau parțială) activităților și a resurselor online în cadrul instituțiilor de educație și a institutelor de cercetare. Această teză prezintă o modalitate de generare automată a cursurilor în cadrul platformei e-learning Moodle, precum și înrolarea automată a studenților, pe baza contractelor de studii. Arhitectura propusă este implementată în cadrul UNSTPB și generează anual 11000 cursuri la care participă aproximativ 35000 utilizatori.

Totodată, se observă și nevoia de eterogenitate a infrastructurilor necesare pentru activitățile de cercetare (platforme multiple, sisteme de operare specifice, recuperare în caz de eroare). Această teză prezintă cum sistemul de operare FreeBSD poate fi integrat într-un mediu cloud, bazat pe OpenStack, pentru a îmbunătăți activitatea dezvoltatorilor și cercetătorilor care folosesc acest sistem de operare în proiectele lor. Mai mult, teza prezintă și îmbunătățirile aduse

algoritmului de migrare a mașinilor virtuale folosind hipervizorul bhyve.

Pe măsură ce atacurile de securitate se înmulțesc, este nevoie ca arhitecturile de tip cluster, Grid și cloud să fie protejate prin intermediul aplicațiilor dedicate. De obicei, se folosesc multiple aplicații complementare de securitate pentru a asigura securitatea rețelei. Sursele atacurilor pot fi atât din exterior (atac asupra serviciilor expuse public), cât și din interior (prin compromiterea utilizatorilor interni). Cu toate acestea, în cadrul dezvoltării aplicațiilor de securitate există problema testării extinse a acestora: rețelele și seturile de date de antrenare a algoritmilor de învățare automată nu sunt suficient de complexe. Această teză propune implementarea unei infrastructuri pilot de testare a aplicațiilor într-un mediu controlat. Mai mult, este propus și un set de date de antrenare (generat prin rularea de atacuri controlate în cadrul unor rețele reale) și testare a aplicațiilor de securitate la nivelul rețelei. Soluțiile propuse au fost implementate pentru testarea SIMARGL, un ansamblu de servicii de securitate, mai precis, a două aplicații de securitate.

# Cuprins

# Capitolul 1

# Introducere

Nevoia de resurse computaționale crește o dată cu evoluția tehnologiei. Problemele complexe de cercetare (simulare proceselor fizice sau chimice, antrenarea modelelor de învățare automată etc.) necesită resurse hardware dedicate. Mai mult, prin intermediul procesului de digitalizare, se migrează spre folosirea resurselor digitale pentru facilitarea lucrului.

În cadrul administrării resurselor de calcul, se observă o tendință de folosire a scalării orizontale (mai multe mașini cu mai puține resurse) în detrimentul scalării verticale (o singură mașină cu multe resurse). Astfel, apar arhitecturile de tip cluster și Grid. Ulterior, prin abstractizarea acestora, se obțin mediile cloud. Cluster este denumirea acordată resurselor de calcul care se află, de obicei, în aceeași locație fizică (centru de calcul) și sunt administrate de aceeași echipă. Prin interconectarea mai multor clustere, se obține Gridul. Fiindcă fiecare cluster este administrat de echipe diferite și conține echipamente diferite, Gridul este un mediu implicit eterogen. Prin noțiunea de "cloud", utilizatorul primește acces la anumite resurse, fără a ști infrastructura de la bază, și plătește doar pentru resursele folosite.

## 1.1 Domeniul tezei

Această teză analizează în detaliu arhitecturile cluster, Grid și cloud prin studierea topologiilor și cerințelor acestora. Teza identifică nevoile acestor medii și propune îmbunătățiri menite să sporească utilizabilitatea și performanța acestora.

Teza se axează pe rețea, soluțiile de tip "middleware" și pe aplicațiile care compun, administrează și rulează în arhitecturi cluster, Grid și cloud. Mai mult, ca studii de caz, analizează arhitectura de cluster și cloud din Universitatea Națională de Știință și Tehnologie Politehnica București (UNSTPB) și arhitectura Grid folosită în cadrul experimentului ALICE al Organizației Europene pentru Cercetări Nucleare (CERN).

Teza identifică o serie de nevoi ale infrastructurilor de calcul și propune soluții care, deși exemplificate în infrastructuri și proiecte specifice, sunt generalizate și pot fi adaptate și altor arhitecturi cluster, Grid și cloud.

## 1.2 Contribuțiile tezei

Pe baza literaturii de specialitate, teza prezintă o vedere generală asupra infrastructurilor de tip cluster care, prin intermediul soluțiilor de tip middleware, se pot interconecta și forma un Grid. În continuare, identifică necesitățile și provocările acestor arhitecturi: existența resurselor eterogene, planificarea și administrarea eficientă a resurselor, securitatea și confidențialitatea datelor, infrastructuri dedicate activităților de educație și cercetare, varietatea seturilor de date, precum și acomodarea diversității aplicațiilor. De asemenea, teza prezintă o analiză detaliată a unui cluster, de la probleme de securitate ce pot să apară la nivelul rețelei până la cerințele de performanță pe care trebuie să le îndeplinească, precum și două studii de caz: arhitectura de calcul (cluster și cloud) din UNSTPB și Grid-ul proiectului ALICE.

Teza prezintă contribuțiile aduse pentru rezolvarea unora dintre problemele identificate în cluster, Grid și cloud. Soluțiile propuse sunt demonstrate în medii specifice (de cercetare).

Teza arată **îmbunătățirile aduse algoritmului de planificare a job-urilor de simulare Monte Carlo**, aplicate în Grid-ul proiectului ALICE. În ALICE, algoritmul Monte Carlo este folosit pentru simularea evenimentelor fizice, în condiții similare cu cele din cadrul coliziunii particulelor în acceleratorul Large Hadron Collider (LHC). Producțiile conțin sute de mii de joburi Monte Carlo, care pornesc din același executabil și de la aceleași condiții inițiale (parametri de intrare), însă care au entropie diferită. În Gridul ALICE, serviciile centrale cer sloturi de câte 24h pe site-uri în care planifică maximal job-urile existente. Fiecărui job îi este atribuită o valoare maximă a timpului de execuție (TTL), pe baza căruia job-ul poate fi planificat într-un slot cu timp suficient de lucru. Job-urile de simulare Monte Carlo necesită un timp foarte mare de lucru din cauza faptului că trebuie să ruleze cu succes și pe hardware-ul mai puțin performant. Astfel, TTL-ul job-urilor de simulare este, în general, mare (aproximativ 20-22h). În consecință, job-urile pot fi planificate numai la începutul rulării unui slot nou pe Grid. Cu toate acestea, s-a observat că din cauza TTL-ului prea mare, site-uri cu hardware performant nu pot planifica acest tip de job-uri, deși timpul lor de execuție este, în realitate, mult mai mic decât TTL-ul cerut. Această teză prezintă o metodă de analiză a factorilor care influențează execuția job-urilor de simulare Monte Carlo în Grid. Analiza este realizată pe informațiile colectate despre datele de rulare a job-urilor din două producții Monte Carlo din Grid (`LHC19_cpubench_pp` și `LHC22e1_extra`).

Pe baza analizei, pe lângă **executabil (care definește întreaga producție)**, au fost identificați și următorii factori care influențează timpul de execuție a job-urilor de simulare Monte Carlo în Grid: modelul de procesor, configurația hyperthreading-ului (activat/dezactivat) și configurația site-ului. Teza propune un **algoritm de estimare a unui TTL maxim calculat dinamic**, pe baza istoricului de rulare a job-urilor împărțite pe baza factorilor care le influențează execuția. În cadrul acestei teze, se folosesc două chei de divizare a setului de date (prima cheie folosește numele producției, numele site-ului, modelul de CPU și configurația de hyperthreading, iar cea de-a doua cheie folosește numele producției, site-ului și nodului de execuție). Pe baza acestor chei, se demonstrează funcționalitatea algoritmului propus și rezultatele obținute atunci când

algoritmul este aplicat pentru producții din cadrul Grid-ului ALICE.

În continuare, teza descrie **îmbunătățirile aduse pentru facilitarea desfășurării activităților de cercetare și învățare**. Aceste îmbunătățiri se axează pe platforme e-learning și pe rezolvarea dificultăților de dezvoltare și cercetare folosind FreeBSD, precum și pe folosirea FreeBSD ca sistem de operare în cluster și cloud.

Moodle este folosit ca platformă e-learning și lucru colaborativ în universități, cât și pentru prezentarea cursurilor de pregătire în firme. Crearea manuală a mii de cursuri și înscrierea a zeci de mii de studenți la acestea este dificilă. Teza propune o metodă de generare automată a cursurilor pe baza unui plan de învățământ și înrolarea studenților la cursuri pe baza contractelor de studii. Îmbunătățirile sunt aplicate în cadrul platformei e-learning Moodle din UNSTPB.

De asemenea, teza arată că sistemul de operare **FreeBSD este necesar în sisteme de tip cloud pentru a facilita desfășurarea activităților de dezvoltare software** (de la compilare la toleranță la defecte). În UNSTPB există un număr mare de proiecte de dezvoltare bazate pe FreeBSD. Astfel, teza arată cum FreeBSD poate fi integrat într-o arhitectură eterogenă. De asemenea, arată modificările aduse sistemului de operare pentru a facilita integrarea în OpenStack, precum și testele realizate.

Mai mult, teza prezintă și **modificările aduse procedurii de migrare a mașinilor virtuale folosind bhyve**, hipervisorul din FreeBSD. Migrarea mașinilor virtuale este o procedură administrativă realizată în cluster pentru echilibrarea încărcării sistemelor sau pentru operații de mentenanță. În teză, sunt detaliate îmbunătățirile aduse migrării mașinilor virtuale în bhyve: adăugarea suportului pentru mașini virtuale cu memorie non-wired (memoria nu este pre-alocată la pornirea mașinii virtuale) și reducerea numărului de operații de copiere ale acelorași date. Pe lângă implementare, teza arată și rezultatele obținute.

În continuare, teza propune o **modalitate de testare a unei colecții de aplicații (toolkit) de securitate**, aplicată în cadrul proiectului SIMARGL. Unele dintre cele mai mari probleme în etapele de dezvoltare și testare a aplicațiilor de securitate este lipsa unei infrastructuri care imită condițiile reale (trafic legitim).

În cadrul tezei, este prezentată **o arhitectură pilot** în care sunt instalate aplicațiile din cadrul toolkit-ului și **integrarea a două rețele reale** (numite `Rețeaua A` și `Rețeaua B` în cadrul tezei), folosite pentru testarea a două dintre aplicațiile din toolkit. Rețeaua A este o rețea mică în interiorul căruia s-a instalat Punch Platform. Această aplicație de securitate era, la momentul realizării testelor, o aplicație matură din punctul de vedere al detecției (antrenată), însă netestată suficient. Folosind Suricata, traficul din Rețeaua A a fost analizat de către Punch Platform. Rețeaua B este o rețea complexă (subrețele multiple, trafic inițiat atât din interior spre exterior, cât și dinspre exterior către serviciile expuse public). În cadrul acestei rețele a fost instalat BDE Engine, o aplicație de securitate care, la momentul respectiv, nu era nici antrenată (cu date reale), nici testată suficient. De asemenea, se prezintă **modalitatea de inserare a două subrețele (una externă și una internă) folosite pentru rularea de atacuri controlate**. Pe

baza atacurilor (scanare, Denial-of-Service, atacuri bazate pe aplicații de tip botnet), a rezultat un **set de date public** ce a fost folosit pentru antrenarea BDE Engine. Mai mult, după faza de antrenare, aplicația a fost testată, iar atacurile au fost detectate cu succes.

## 1.3 Structura tezei

Această teză este structurată astfel:

1. Primul capitol evidențiază domeniul tezei și obiectivele acesteia.
2. Al doilea capitol prezintă diferențele dintre cluster, Grid și cloud, detaliind arhitectura acestora, cu accent pe problemele de securitate și performanță ce pot să apară, dar și pe nevoile existente. Tot în acest capitol sunt prezentate două studii de caz: clusterul și cloud-ul din UNSTPB și Grid-ul experimentului ALICE.
3. Capitolul trei arată studiul realizat asupra factorilor care influențează execuția producțiilor Monte Carlo în Grid-ul ALICE. Se prezintă un algoritm de estimare a unei valori maxime pentu un job, pe baza istoricul de rulare a job-urilor similare. Algoritmul propus este menit să reducă TTL-ul unui job, îmbunătățind procesul de planificare a job-urilor. De asemenea, capitolul arată și rezultatele experimentale obținute prin implementarea algoritmului și testarea acestuia pe date extrase din Grid-ul experimentului ALICE.
4. Al patrulea capitol prezintă îmbunătățirile aduse pentru facilitarea activităților de cercetare și învățare. Descrie operațiile automate de generare a cursurilor și înrolare a studenților, cu aplicare în platforma e-learning a UNSTPB, Moodle. În continuare, teza descrie cum FreeBSD este utilizat în cadrul procesor de cercetare în UNSTPB și cum FreeBSD poate fi folosit într-o infrastructură cluster și cloud, precum și îmbunătățirile aduse domeniului.
5. Capitolul cinci propune o metodologie de testare a unui toolkit de securitate într-un mediu real. Metodologia a fost aplicată în cadrul proiectului SIMARGL. Capitolul prezintă, de asemenea, și modalitatea de testarea a două soluții de securitate bazate pe algoritmi de învățare automată prin introducerea acestora în două rețele monitorizare. Mai mult, capitolul descrie și atacurile controlate conduse în cadrul rețelelor pentru antrenarea (prin generarea unui set de date public) și, respectiv, testarea unor aplicații de securitate.
6. Ultimul capitol prezintă concluziile tezei.

# Capitolul 2

# Infrastructuri cluster, Grid și cloud

Pe baza literaturii de specialitate, acest capitol prezintă o privire de ansamblu asupra mediilor cluster, Grid și cloud, axându-se pe lipsurile și necesitățile curente ale acestora. Sunt prezentate și două studii de caz, evidențiind structuri reale ale unui cluster și cloud instituțional (UNSTPB) și a unui Grid folosit în cercetare (ALICE).

## 2.1 Topologia generală a unei infrastructuri cluster și Grid

Cluster, Grid și cloud [1, 2, 3] sunt trei termeni care descriu arhitectura dedicată (hardware și software) utilizată în cadrul infrastructurilor mari pentru calcul sau stocare [2, 4, 5, 6, 7]. Potrivit literaturii [8, 6], se observă o tendință spre eterogenizarea arhitecturilor, atât hardware (CPU, GPU etc.), cât și software (sisteme de operare, hypervisoare).

Noțiunea de cluster se referă, în mod curent, la resursele de calcul dintr-o singură încăpere, cu o singură echipă de administratori, în timp ce Grid-ul este distribuit geografic și format din mai multe clustere. Nivelul de eterogenitate al Grid-ului este mult mai mare decât cel al clusterului [8, 9]. Cloud-ul [1, 2, 6, 8] este un mediu care abstractizează accesul la resurse (din punctul de vedere al utilizatorului), fie ele grupate în cluster sau Grid. Prin intermediul cloud-ului, utilizatorii primesc acces doar la resursele de care au nevoie (infrastructură, platformă sau aplicație) și pot scala în funcție de nevoie și costuri.

Pe baza literaturii de specialitate [1, 2, 3, 6, 8, 9], Figura 2.1 detaliază o arhitectură generală a unui cluster și a unui Grid. Orizontal, sunt două niveluri: hardware și software. Vertical, sunt prezentate cele patru componente principale dintr-un cluster: stocarea, serverele pentru calcul, rețelistica și orchestrarea resurselor. Fiecare din aceste componente folosește hardware și software dedicat. În aceste arhitecturi, virtualizarea (containere sau mașini virtuale), folosită pentru izolare și limitare a resurselor, și soluțiile de tip "middleware" (accesul la resurse, planificarea și monitorizarea job-urilor) joacă un rol foarte important.

Figura 2.1 arată, de asemenea, structura unui Grid care este format din mai multe clustere, interconectarea lor făcându-se prin intermediul rețelei publice.

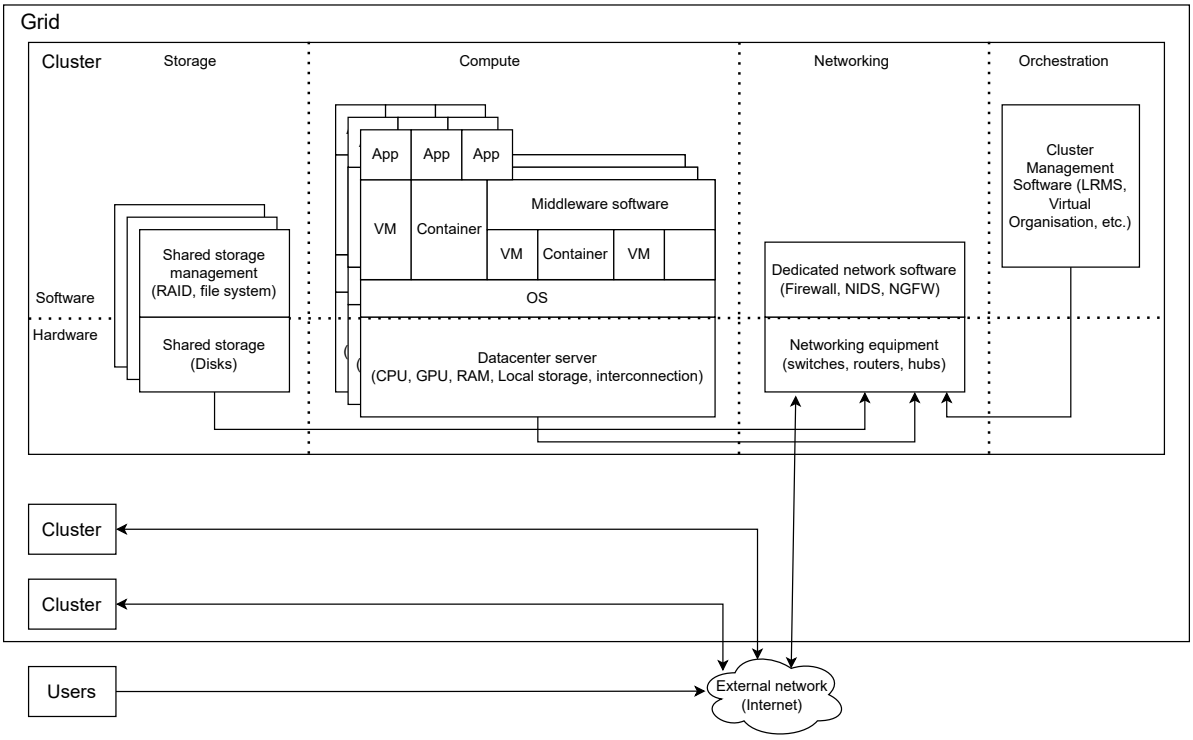



Figura 2.1: Privire de ansamblu asupra arhitecturii unui cluster și a unui Grid

## 2.2 Nevoi în cluster, Grid și cloud

Conform literaturii de specialitate, am identificat următoarele categorii de nevoi în medii cluster, Grid și cloud:

- **nevoia de resurse eterogene**: problemele complexe (biobanking [10], big data [11], procesare de date [5, 7, 12, 13]) au nevoie de multe resurse pentru a rula. Deși unele arhitecturi sunt omogene [14], tendința este de a genera infrastructuri eterogene [4, 5, 6, 8, 15] prin integrarea harware-ului nou achiziționat în infrastructurile curente.
- **planificare și administrare a resurselor**: planificarea corectă și eficientă a job-urilor [1, 3, 16, 17, 18] rămâne una dintre cele mai dificile probleme în cadrul clusterelor și Grid-urilor (în unele studii [19], planificarea fiind considerată o problemă NP-hard).
- **infrastructuri eficiente din punctul de vedere al costurilor sau al energiei consumate** [3, 15, 17]: deoarece multe modele optează pentru servicii în care utilizatorii plătesc doar pentru resursele utilizate, este nevoie ca arhitecturile să fie eficiente din punctul de vedere al costurilor. Mai mult, nodurile de cluster și Grid care nu rulează (sunt în starea idle), consumă energie [17]. O soluție pentru reducerea costurilor [15] poate fi migrarea eficientă a mașinilor virtuale.
- **securitate și confidențialitate**: topologiile complexe cu mulți utilizatori sunt supuse amenințărilor de securitate [20, 21, 22, 23, 24, 25] prin atacuri de tip Denial of Service, man in the middle, minare, scanare, phising sau botnet.
- **facilitatea activităților de cercetare și educație** [10, 11, 18, 25, 26, 27]: infrastructura

IT este vitală pentru îmbunătățirea eficienței profesorilor și a studenților [5, 28, 29, 30, 31, 32, 33, 34], iar flexibilitatea infrastructurilor cloud facilitează activități de cercetare și inovare [4, 25, 35].

- **varietatea resurselor și a seturilor de date (datasets)**[14, 20, 36, 37]: în cadrul testării algoritmilor propuși, cercetătorii au nevoie de seturi de date variate pentru a acoperi un număr cât mai mare de cazuri.
- **arhitecturi în care se pot rula cu ușurință o gamă largă de aplicații**[25, 29, 35, 38]: pentru facilitatea proceselor de învățare.

Detaliind securitatea arhitecturilor cluster și Grid, atacurile pot viza o singură componentă sau pot să apară sub forma unui vector. Principalii pași în cadrul unui vector de atac sunt următorii [39, 40]: **recunoaștere**, prin scanarea IP-urilor, porturilor deschise, protocoalelor sau resurselor umane; **scanarea vulnerabilităților**; **utilizarea vulnerabilitătilor** pentru a pătrunde în rețea; **descoperirea topologiei rețelei interne**, bazată pe pașii prezentați în faza de recunoaștere; atacuri către țintele vizate.

Pe baza literaturii de specialitate [20, 21, 22, 23, 24, 37, 41, 42, 43, 44], următoarele tipuri de atacuri pot fi folosite în cadrul unei rețele:

- atacuri de tip **(Distibuted) Denial of Service (DDoS)** [20, 21, 22, 23, 37, 40, 41, 42, 44] - acestea pot să apară la toate nivelurile (infrastructură, rețea, sistem de operare, middleware, aplicație) și fac acea componentă atacată inutilizabilă.
- atacuri de tip **Man-in-the-Middle (MITM)** [22, 37, 45] - acestea apar atunci când un actor malițios (atacatorul) intervine în comunicația între două entități și o alterează pe baza intențiilor ulterioare.
- atacuri bazate pe recunoaștere, scanare sau colectarea datelor [22, 37] - folosite pentru a crea o vedere generică asupra rețelei și a punctelor vulnerabile din aceasta.
- atacuri de **minare de cryptomonede** [46, 47] - utilizate pentru câștiguri materiale (cryptomonede), acestea consumă multă putere de calcul.
- atacuri de tip **phishing** [37, 43] - prin păcălirea utilizatorilor să acceseze, descarce sau ruleze diverse link-uri sau fișiere, atacatorii pot să fure date personale (parole, date bancare) sau să intre în rețeaua internă.
- atacuri bazate pe **ransomware** [43] - combinate cu alte tipuri de atacuri (ex., phising), instalează software malițios pe sistemele victimei.
- atacuri de **escalare a privilegiilor** [42] - folosite pentru a executa comenzi ca un utilizator privilegiat.
- atacuri de tip **botnet** [21, 37, 40] - folosite pentru extinderea rețelei de sisteme infectate. Atunci când rețeaua este suficient extinsă, sistemele centrale (Command & Control) pot lansa atacuri distribuite.

Soluțiile de mitigare a atacurilor presupun atât mărirea resurselor [44], cât și limitarea ratei de transfer în rețea sau folosirea mașinilor virtuale pentru izolarea serviciilor ce pot fi atacate [22]. Cu toate acestea, este nevoie de aplicații dedicate pentru detecția anomaliilor la nivel de rețea. Aceste soluții sunt, în general, complementare [22, 24, 48]. Deoarece unele atacuri sunt greu de detectat [22, 40, 41], tendința este de folosire a inteligenței artificiale și a metodelor de învățare automată pentru protejarea rețelei [23, 36, 37, 40, 44, 47]. Totuși, un neajuns întâlnit în procesul de dezvoltare al acestor aplicații este lipsa unei infrastructuri complexe și a traficului real îmbinat cu trafic malițios pentru testarea modelului.

## 2.3 Studiu de caz: Arhitectura de calcul în UNSTPB

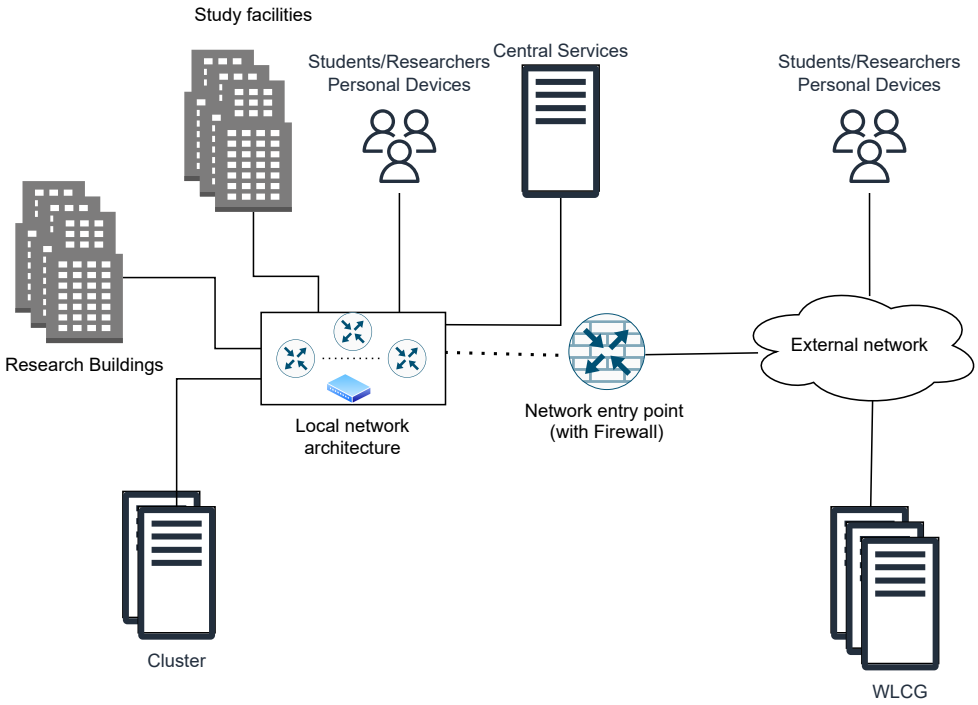



Figura 2.2: Vedere de ansamblu a arhitecturii UNSTPB

Figura 2.2 prezintă o privire de ansamblu a arhitecturii de rețea din cadrul UNSTPB și este formată din următoarele componente: **servicii centrale** de administrare a identităților și serviciilor (LDAP, DNS, echipamente dedicate, SSO), **clădiri** folosite în scop de cercetare și educație (laboratoare, săli de curs, birouri, bibliotecă) și conectate la rețeaua internă, **clusterul** cu resurse eterogene [4, 24] folosite pentru activități didactice și de cercetare și care face parte din WLCG (Worldwide LHC Computing Grid) contribuind în cadrul proiectului ALICE, și **dispozitivele personale** care se conectează la rețea (WiFi sau ethernet).

În cadrul WLCG, clusterul UPB pune la dispoziție experimentului ALICE 6PB storage, 150 noduri de calcul care însumează aproximativ 2400 core-uri și 10TB RAM. Arhitectura clusterului este una eterogenă, cu cel puțin 8 modele diferite de CPU (Intex(R) Xeon(R) E5 și Intel(R) Gold) folosite în Gridul ALICE și alte modele de CPU (AMD Opteron, AMD Quad, Intel(R) Xeon(R)) și GPU-uri puse la dispoziție studenților și cercetătorilor. Ca soluții de orchestrare,

UPB folosește SLURM, OpenStack și Microsoft Hyper-V.

Pentru buna funcționare a activităților din interiorul universității, UPB gestionează servicii de administrare (LDAP, DNS, mail, SSO, VPN) și de e-learning (Moodle, OpenStack, wiki, site-uri generice). Aceste servicii sunt protejate de mai multe soluții de securitate (firewall, soluții dedicate, configurări de separare a rețelelor prin intermediul vlan-urilor).

În cadrul centrului universitar UPB există 15 facultăți care însumează mai mult de 35000 de profesori, studenți și personal administrativ. Fiecare facultate are propriul plan de învățământ, generându-se anual 11000 de cursuri în plaforma de e-learning a UPB, bazată pe Moodle.

## 2.4 Studiu de caz: Arhitectura distribuită de calcul din Grid-ul ALICE

Figura 2.3 prezintă o vedere de ansamblu a Grid-ului ALICE și interacțiunea dintre componentele ei [12, 13, 49, 50, 51, 52, 53]. Arhitectura are trei componente principale: serviciile centrale (**Central Services**) care administrează și monitorizează activitatea din Grid, site-urile din Grid care pun la dispoziție experimentului resursele de calcul, și utilizatorii care submit job-uri în Grid.

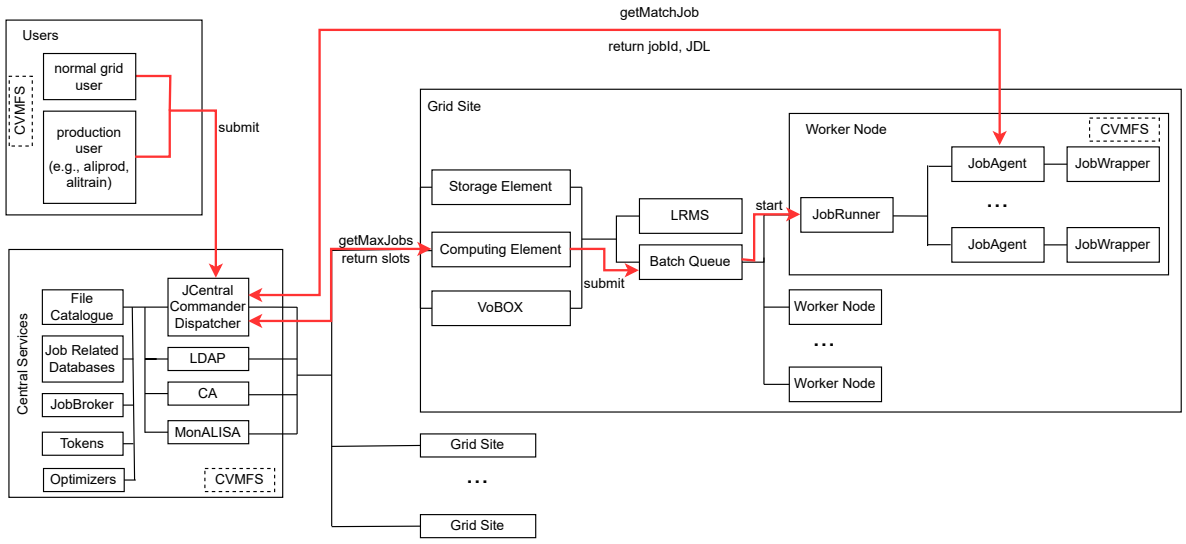



Figura 2.3: ALICE Grid Architecture

**Serviciile centrale** rulează la CERN și au, la rândul lor, mai multe componente: **JCentral**, folosit ca punct de comunicare și lansare de comenzi (API); **File Catalogue**, un modul care realizează o mapare între numele logic asociat fișierelor din Grid și locația efectivă a acestora; baze de date care conțin informații despre job-uri, folosite pentru administrarea și planificarea acestora; **JobBroker**, modulul responsabil de planificare job-urilor; module de optimizare a diverselor operații legate de planificarea job-urilor; servicii care fac managementul certificatelor X.509 folosite pentru comunicarea sigură în Grid; un server **LDAP** folosit pentru reținerea informațiilor despre utilizatori și site-uri; un **CA** (Certificate Authority) care semnează și va-

lidează certificatele X.509 folosite pentru autentificare, autorizare și acces; **MonALISA**, un serviciu de monitorizare și contabilitate a execuției job-urilor din Grid; **CVMFS (CERNVM File System**, un sistem de fișiere distribuit prin intermediul căruia se publică executabilele folosite în Grid.

**Utilizatorii** din Grid-ul ALICE sunt de două tipuri: utilizatori individuali, cercetători asociați experimentului care trimit job-uri personalizate pentru execuția în Grid, și conturi de producție, prin intermediul cărora se trimit job-uri complexe de simulare, analiză și reconstrucție, sub forma unor producții. O producție este o clasă de job-uri de același tip care trebuie execute în Grid. Fiecare producție are asociat un tag unic pentru identificare și un fișier de configurare (JDL - Job Description Language) în care se găsesc specificațiile job-urilor. Prin intermediul unei producții, se lansează multiple job-uri similare (diferențierea se face prin parametrii primiți de job).

Fiecărui job îi este asociată o stare pe toată durata de viață. Câteva dintre cele mai importante stări sunt următoarele: **INSERTED** - job-ul a fost submis în Grid și salvat în baza de date de job-uri, **WAITING** - job-ul așteaptă să fie planificat, **ASSIGNED** - job-ul a fost planificat pe un nod din Grid, **STARTED** - job-ul a fost lansat în execuție pe nodul din Grid, **RUNNING** - job-ul rulează în Grid, **DONE** - job-ul și-a încheiat execuția cu succes, **ERROR_*** - mai multe coduri de eroare care indică de ce a eșuat execuția unui job din Grid.

Grid-ul ALICE este format din aproximativ 60 de **site**-uri care își pun la dispoziție resursele de calcul. Fiecare site este compus din următoarele elemente: **Storage Element (SE)**, un modul care administrează spațiul de stocare de pe site; **Computing Element (CE)**, un modul ce administrează activitățile de calcul de pe site, fiind o interfață între serviciile centrale și Local Resource Management System (LRMS); **VoBOX (Virtual Organization BOX)** [50, 54], un serviciu care administrează contabilitatea și monitorizarea activităților de pe site; **Local Resource Management System (LRMS)**, un modul care administrează resursele disponibile, precum și nodurile de execuție din grid; **Worker Nodes (WN)**, nodurile din Grid care execută job-uri.

CE trimite cereri către planificatorul local de job-uri (LRMS), iar nodul de execuție pornește un **JobRunner**, un modul de administrare a job-urilor care rulează pe WN [12]. Fiecare **JobRunner** primește un certificat valabil 24h, folosit pentru comunicarea cu serviciile centrale și, pentru fiecare job primit de la serviciile centrale pentru execuție, pornește câte un **JobAgent**. Acesta din urmă, prin intermediul **JobWrapper**, configurează mediul de lucru și pornește execuția job-ului. Durata de viață a certificatului este durata de viață a unui slot cerut pe un site al Grid-ului. Deoarece un site poate face parte din mai multe organizații virtuale (VO), slotul cerut trebuie folosit eficient, încercând să se planifice un număr cât mai mare de job-uri într-un singur slot.

SiteSonar [55, 56] este un modul prin intermediul căruia se interoghează configurările și resursele existente pe un nod din Grid. SiteSonar execută, o dată pe zi, o serie de scripturi care extrag informații despre host-ul de execuție. Informațiile colectate sunt trimise către serviciile

centrale.

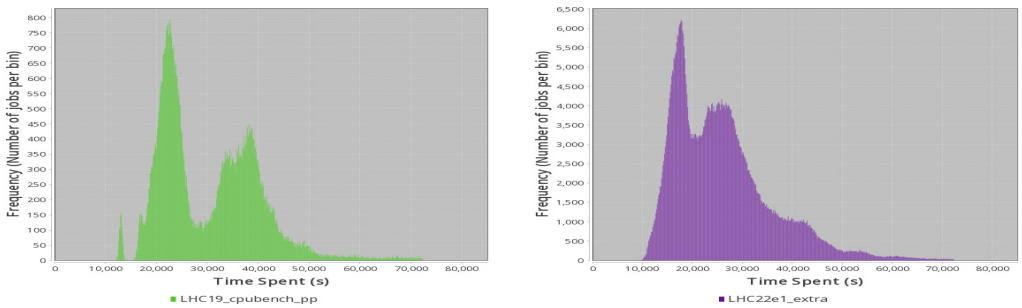
Tabela 2.1: Statistici despre activitatea Grid-ului ALICE în decurs de o săptămână, generat cu date extrase din SiteSonar.

| **Hosturi** | **Site-uri** | **Sisteme de operare** | **Modele de CPU** |
|---|---|---|---|
| 11676 | 55 | 15 | 131 |

Tabelul 2.1 prezintă statisticile extrase prin analiza datelor din SiteSonar într-o săptămână. Acest tabel prezintă eterogenitatea ridicată a Grid-ului ALICE.

Prin intermediul acțiunilor de monitorizare a Grid-ului, s-a observat faptul că, deși anumite slot-uri deschise pe site-uri aveau o durată de viață rămasă de peste jumătate din certificatul lor, instanțele **JobAgent** nu primeau job-uri spre execuție. Acest lucru se întâmplă deoarece majoritatea job-urilor cer, prin intermediul TTL-ului (timp maxim de execuție), aproape tot intervalul de timp care poate fi atribuit unui job (ex. 72000 sau 80000 secunde, echivalentul a 20 sau aproape 22 de ore). Cu toate acestea, majoritatea job-urilor termină execuția mult sub TTL-ul cerut. Din perspectiva Grid-ului, reducerea TTL-ului nu este fezabilă: TTL-ul este setat astfel încât să acomodeze toate configurațiile din Grid, iar job-ul să termine execuția și pe cel mai lent nod de execuție.

Figura 2.4 arată distribuția timpului petrecut de job-urile de simulare Monte Carlo din două producții din Grid-ul ALICE. În figură, se poate observa faptul că job-urile petrec între 10000 și 720000 secunde în execuție. Astfel, TTL-ul nu poate fi redus empiric deoarece ar însemna că o parte din job-uri ar fi omorâte. Astfel, este nevoie de o estimare dinamică a TTL-ului, pe baza istoricului de execuție a job-urilor similare. Această teză se axează pe producțiile de simulare Monte Carlo (procese intensive computațional).




(a) LHC19_cpubench_pp (b) LHC22e1_extra

Figura 2.4: Timpul pretrecut în execuție de job-urile de simulare Monte Carlo în două producții din Grid-ul ALICE. Valorile timpului petrecut sunt între 10000 și 720000, cea din urmă fiind valoarea TTL-ului cerut de către job-uri.

## 2.5 Îmbunătățirile propuse pentru arhitecturile de calcul

Pe baza nevoilor identificate în cadrul acestui capitol, această teză propune soluții pentru îmbunătățirea:

- **mecanismului de planificare în medii Grid** prin implementarea unui algoritm de estimare a timpului maxim de execuție a unui job, pe baza istoricului de execuție. Soluția propusă este exemplificată în cadrul Grid-ului ALICE, însă poate fi adaptată și în alte Grid-uri.
- **infrastructurii și suportului pentru activități de cercetare și învățare** prin îmbunătățirile aduse procesului de generare automată de cursuri și înrolări în platforme e-learning și prin îmbunătățirea modului de folosire al FreeBSD în mediul cloud. Soluțiile propuse sunt demonstrate în cadrul arhitecturii cluster și cloud din UNSTPB.
- **eficientizarea consumului energetic** prin îmbunătățirile aduse procesului de migrare live a mașinilor virtuale folosind bhyve. Așa cum este demonstrat în [15], migrarea live a mașinilor virtuale poate fi folosită pentru balansarea încărcării nodurilor, ducând astfel la clustere eficiente energetic.
- **securității** prin adăugarea de îmbunătățiri aduse hypervisorului bhyve, dar și prin implementarea de rețele pilot de testare în medii reale a aplicațiilor de securitate.
- **varietății seturilor de date** prin crearea de noi seturi de date cu fluxurile de trafic de rețea obținute în cadrul rulării controlate a unor atacuri de rețea într-un mediu real.

# Capitolul 3

# Îmbunătățiri aduse în planificarea job-urilor de simulare MonteCarlo în Griduri eterogene

Primul pas în optimizarea planificării job-urilor de simulare Monte Carlo prin estimarea unei valori a TTL-ului pe baza istoricului de execuție a job-urilor în Grid este studierea producțiilor existente și determinarea factorilor care influențează durata de execuție a job-urilor. În acest capitol este prezentată analiza producțiilor Monte Carlo din cadrul Grid-ului ALICE. Pe baza factorilor identificați și a unui algoritm de estimare propus, se estimează TTL-ul și se prezintă rezultatele obținute atunci când algoritmul este aplicat pe date reale extrase din Grid-ul ALICE.

## 3.1 Analiza duratei job-urilor de tip MonteCarlo în Griduri eterogene

Fiecare producție Monte Carlo repetă același proces de simulare, în aceleași condiții ca cele stabilite în cadrul experimentelor fizice, schimbând doar sursa de entropie pentru generarea părții aleatoare din proces. Din perspectiva Grid-ului, job-urile de simulare sunt computațional intensive (CPU-intensive), iar operațiile I/O sunt minimale și nu influențează timpul de execuție. În cazul analizei, au fost considerați următorii factori:

- **configurația hardware și procesorul** - aplicațiile CPU-intensive sunt influențate în mod direct de procesor și de hardware.
- **configurația software și activitatea site-ului** - chiar dacă două site-uri au hardware identic, soluțiile software (sisteme de operare, aplicații middleware, supra-folosire (overprovisioning) a resurselor) alese pot influența execuția job-urilor. Mai mult, site-urile pot face parte din mai multe organizații virtuale (VO) și două job-uri din organizații diferite pot să ruleze în paralel, împărțind aceleași resurse fizice.

- **caracteristici ale producției**.

Eșantioanele de date folosite în cadrul analizei sunt compuse din informațiile extrase despre două producții Monte Carlo (`LHC19_cpubench_pp` and `LHC22e1_extra`). Pentru aceste producții, au fost extrase informații despre job-urile care au terminat execuția cu succes (în starea **DONE**). Din informațiile de log furnizate de job, au fost extrase site-ul și nodul de execuție, precum și timpul total de rulare. Prin intermediul SiteSonar, pe baza numelui host-ului, s-au extras configurația procesorului (nume, familie, frecvență, core-uri, hyperthreading), planificatorul local (HTCondor, Slurm), soluțiile de containerizare (Docker, Singularity/Apptainer, fără containerizare), virtualizarea, vulnerabilitățile la nivel de unitate de execuție (Spectre/Meltdown), precum și mitigarea acestora (dacă există). Toate datele din această teză sunt anonimizate. Tabelul 3.1 arată dimensiunea seturilor de date folosite în cadrul analizei.

Tabela 3.1: Seturile de date folosite în cadrul analizei

| **Producție** | **Data și numărul de job-uri** | | |
|---|---|---|---|
| **LHC19_cpubench_pp** | **Iun 2022** 77623 | **Iul 2022** 82733 | **Oct 2022** 858532 |
| **LHC22e1_extra** | **Oct-Noi 2022** 634088 | | **Apr 2023** 66930 |

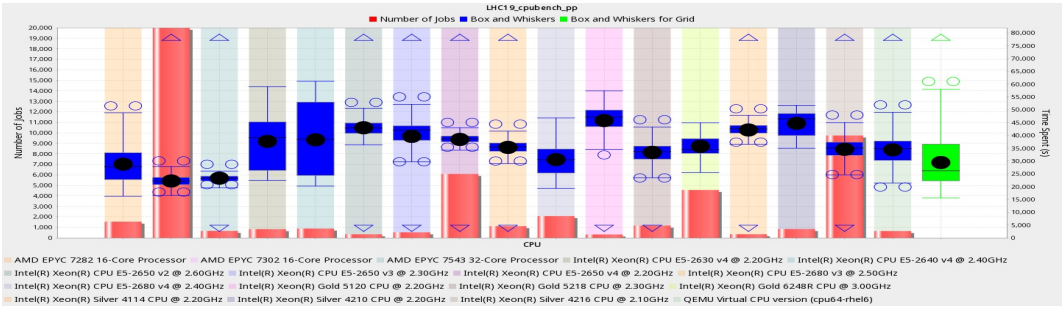
Pentru analiza seturilor de date, au fost folosite **histograme** [1], scopul fiind de a determina forma distribuției; și reprezentarea **Box & Whiskers** [2], pentru reprezentarea, pe același grafic a mai multor subseturi de date, cvartilele în care se află fiecare 25% din date, mediana (linia orizontală), media (punctul negru), outlier-ele (cercurile goale; triunghiurile din grafic arată outlier-ele depărtate). Studiul are ca scop determinarea factorilor pe baza cărora subseturile obținute au o distribuție normală, cu puține outlier-e, cu cvartile compacte (set restrâns de valori). În graficele prezentate în acest capitol, 'Time spent' reprezintă timpul de execuție al job-ului în secunde, iar 'H-ON'/'H-OFF' reprezintă configurația pentru hyperthreading.

Analiza prezentată în următoarele paragrafe a ținut cont de modelul de procesor, configurația de hyperthreading (conform literaturii [57, 58, 59, 60], activarea hyperthreading-ului scade performanța job-urilor de tip Monte Carlo) și configurația site-ului și a nodurilor de execuție. În cadrul studiului, s-a pornit de la premisa că site-urile sunt, de cele mai multe ori, configurate similar prin metode automate de configurare precum Puppet sau Ansible.

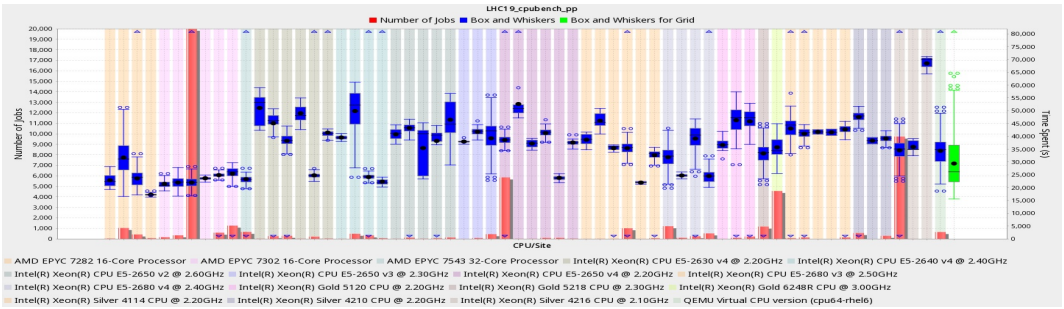
Figurile 3.1 și 3.2 arată reprezentarea box & whisker pentru durata job-urilor împărțite după procesor (3.1a, 3.2a) și după site și procesor (3.1b, 3.2b). În Figurile 3.1b și 3.2b sunt reprezentate aceleași modele de procesor ca în Figurile 3.1a și 3.2a, de data aceasta seturile de date fiind împărțite și după site (se păstrează ordinea si culoarea). Din cauza numărului mare de

[1] Reprezentarea prin histograme, Online: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/data-presentation/histograms.html, Accesat 25 Iulie 2023

[2] Reprezentarea prin Box & Whiskers, Online: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/data-presentation/box-and-whisker-plots.html, Accesat 25 Iulie 2023




(a) Producție/CPU




(b) Producție/Site/CPU

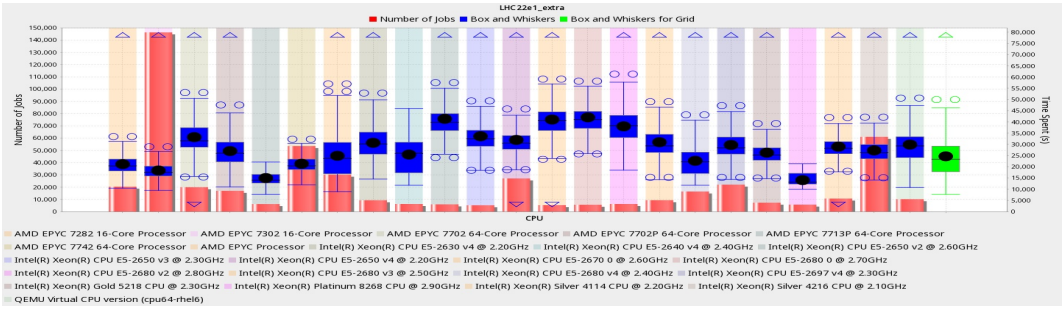
Figura 3.1: LHC19_cpubench_pp: Reprezentare Box & Whiskers pentru seturile de date împărțite după *producție/CPU* și, respectiv, *producție/site/CPU*. Seturile de date sunt în ordine alfabetică, sortate pe baza numelui procesorului. Bara roșie reprezintă numărul de job-uri din subset (axa y din stâga), dreptunghiurile albastre reprezintă durata job-ului (axa y din dreapta), în timp ce dreptunghiul verde reprezintă activitatea Grid-ului). Fiecare Model de Procedor are o culoare diferită și este expandat per site în figura (b). Cercurile goale reprezintă outliere, iar triunghiurile arată că setul de date are outliere distante.

procesoare, în aceste grafice sunt prezentate doar o parte dintre rezultate. Restul reprezentărilor sunt similare cu cele prezentate. Figurile arată că dreptunghiurile (box, reprezentate cu albastru) sunt mai înguste când site-ul este considerat, iar același model de procesor nu are mereu comportament uniform pe site-uri diferite.

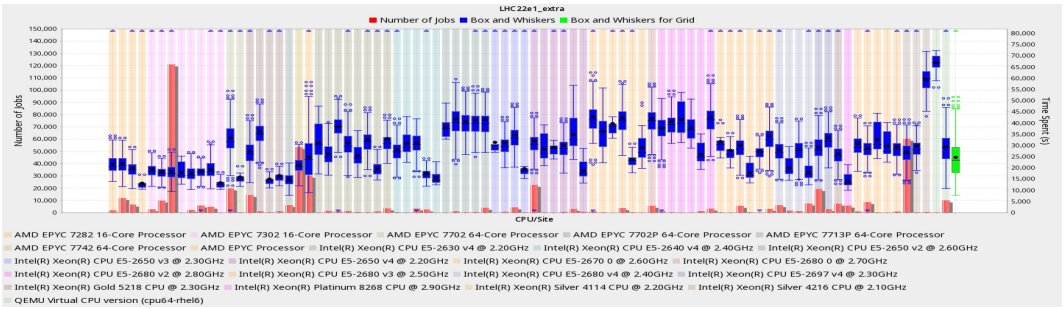
Figura 3.3 prezintă influența configurației de hyperthreading în execuția job-urilor de tip Monte Carlo. Din grafice, se observă faptul că activarea hyperthreading-ul duce la creșterea timpului de execuție a job-urilor de simulare Monte Carlo. Comportamentul este similar cu cel descris în literatura de specialitate [57, 58, 59, 60]. În general, hyperthreading-ul este folosit și ajută în cazul job-urilor I/O intensive.

Figurile 3.4 și 3.5 arată comportamentul duratei de execuție a job-urilor grupate după CPU și configurație de hyperthreading per fiecare site și, respectiv, grupate după site per CPU și configurație de hyperthreading. Astfel, se observă că atunci când job-urile sunt împărțite după producție, site, modelul de procesor și configurația de hyperthreading, seturile de date obținute prezintă distribuție normală.

Figura 3.5 arată cum durata de execuției a job-urilor pe același model de procesor diferă în funcție de site. Chiar dacă pentru unele site-uri și modele, distribuțiile se suprapun, pentru altele, nu. Există și cazuri aparte în care pentru un site și un model de procesor, distribuția este




(a) Producție/CPU




(b) Producte/Site/CPU

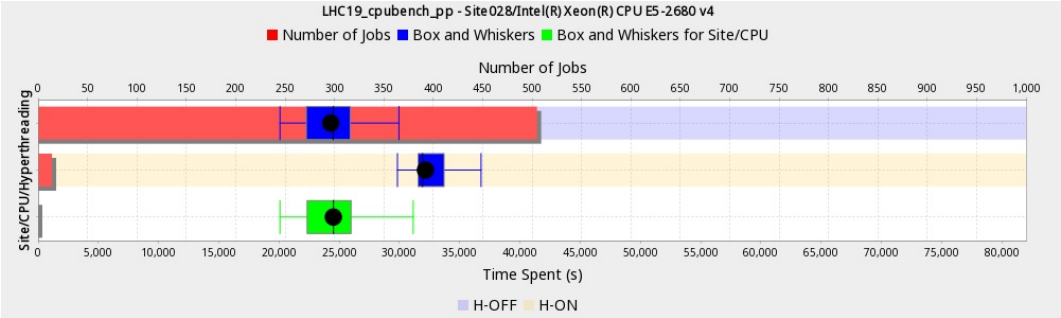
Figura 3.2: LHC22e1_extra: Reprezentare Box & Whiskers pentru seturile de date împărțite după *producție/CPU* și, respectiv, *producție/site/CPU*. Seturile de date sunt în ordine alfabetică, sortate pe baza numelui procesorului. Bara roșie reprezintă numărul de job-uri din subset (axa y din stâga), dreptunghiurile albastre reprezintă durata job-ului (axa y din dreapta), în timp ce dreptunghiul verde reprezintă activitatea Grid-ului). Fiecare Model de Procedor are o culoare diferită și este expandat per site în figura (b). Cercurile goale reprezintă outliere, iar triunghiurile arată că setul de date are outliere distante.

foarte lată, iar în altele, apar distribuții bi-modale.

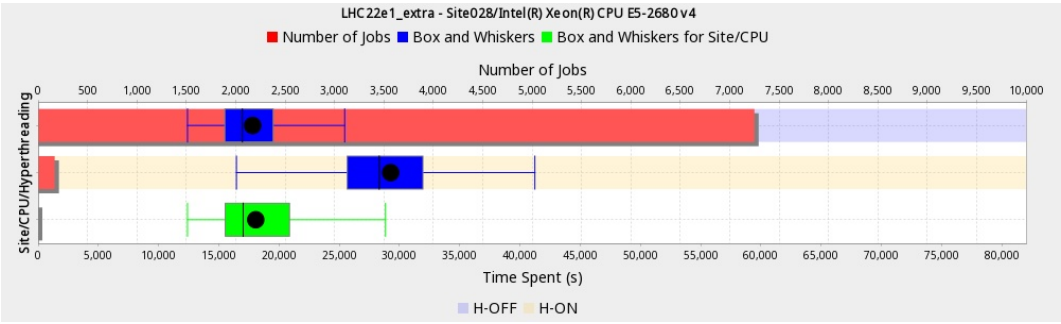
Pe baza analizei efectuate, s-a observat că setul de date, împărțit pe baza celor patru factori: producția (executabilul folosit în simulare și argumentele acestuia), site-ul (cu aceeași configurație pentru aproape toate host-urile), modelul de procesor și configurația de hyperthreading, majoritatea subseturilor obținute au distribuție normală. Cu toate acestea, există câteva cazuri particulare în care subsetul obținut are distribuție bi-modală (Figurile 3.4 și 3.5 pentru `LHC22e1_extra`), are în continuare un interval foarte lat de valori ale timpului de execuție, sau are distribuție deformată (skewed). De asemenea, toate seturile de date obținute au valori anormale (outliere).

Cazurile anormale întâlnite au fost analizate separat pentru a determina dacă există și alți factori care influențează timpul de execuție a job-urilor, alții decât cei considerați până acum. Au fost luate în considerare aplicațiile de administrare locală a job-urilor pe site, virtualizarea, soluția de containerizare, precum și alte aspecte hardware (vendorul de hardware, RAM-ul, configurația NUMA a site-urilor etc.).

În urma analizei, nu există un alt factor care singur să influențeze execuția, ci execuția este influențată de un cumul de facturi. Însă, aceste cazuri sunt, de obicei izolate, iar numărul de posibile combinații existente este prea mare pentru a fi luat în considerare pentru estimarea




(a) LHC19_cpubench_pp




(b) LHC22e1_extra

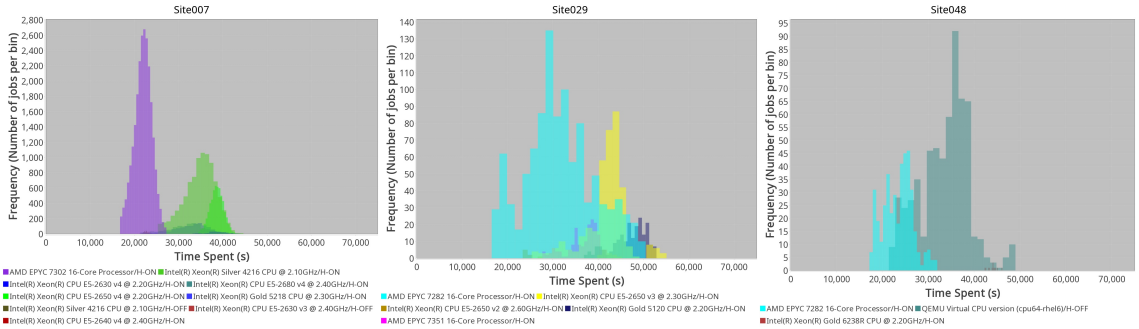
Figura 3.3: Box & Whiskers pentru Site028. Dreptunghiul roșu reprezintă numărul de job-uri pentru fiecare subset (axa x de sus), dreptunghiurile albastre reprezintă box & whisker pentru timpul petrecut de job-uri împărțite după configurația de hyperthreading, iar dreptunghiul verde prezintă rezultatele per Site. `H-ON` înseamnă că hyperthreading-ul este activat, iar `H-OFF` înseamnă că hyperthreading-ul este dezactivat

TTL-ului. De asemenea, se poate considera că nodurile de execuție au configurații similare, realizate prin intermediul aplicațiilor de administrare a configurării, precum Puppet și Ansible. Astfel, numele site-ului acoperă majoritatea configurațiilor urmărite în cadrul acestei analize.

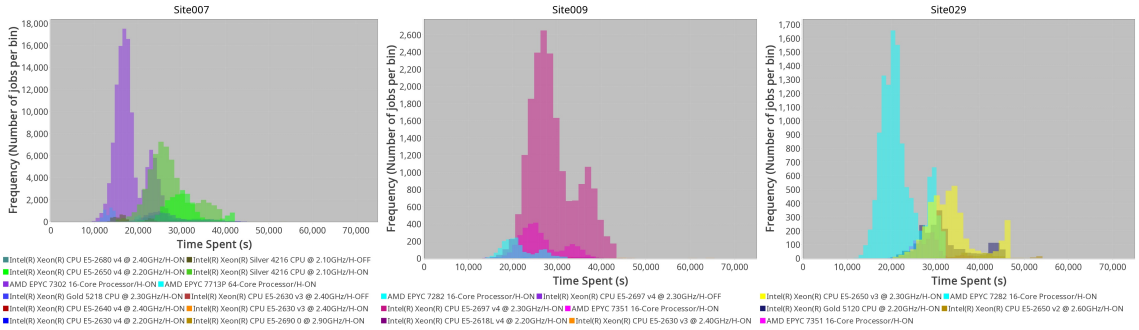
O altă analiză efectuată pe seturile de date este legată de comportamentul producției în timp. Pe baza analizei, există cazuri izolate când, la diferență de 5-6 luni de la rularea unei producții Monte Carlo, comportamentul acesteia se schimbă. În cazul analizat, parametrii producției s-au schimbat, dublându-se numărul de evenimente generate. În consecință, durata de execuție a unui job s-a dublat. Aceste cazuri izolate indică faptul că producțiile sunt dinamice, iar rularea aceleiași producții la intervale de timp mari poate însemna și schimbarea parametrilor de execuție.

Pe baza analizei din această secțiune, se pot trage următoarele concluzii:

- site-urile sunt configurate pentru o performanță bună (ex., hyperthreading-ul este activat deoarece îmbunătățește performanța job-urilor I/O).
- Modelul de procesor și configurația de hyperthreading influențează timpul de execuție a job-urilor.
- configurațiile la nivel de site depind de administratorii de sistem, iar două site-uri din Grid nu sunt configurate identic.




(a) LHC19_cpubench_pp/Site007 (b) LHC19_cpubench_pp/Site029 (c) LHC19_cpubench_pp/Site048




(d) LHC22e1_extra/Site007 (e) LHC22e1_extra/Site009 (f) LHC22e1_extra/Site029

Figura 3.4: Distribuțiile duratei de execuție a job-urilor, împărțite după CPU și configurație de hyperthreading și grupate după numele site-ului.

- același model de CPU se poate comporta diferit în funcție de configurația site-ului.
- parametrii producției influențează durata de execuție a job-ului (ex. numărul de evenimente generate), iar executabilele pot avea particularități (distribuții bi-modale sau oblice). Trebuie menționat că numărul de evenimente dintr-o producție nu poate fi determinat înainte de rularea job-ului.
- comportamentul producțiilor este dinamic: unele producții au același comportament în momente diferite de timp, altele au comportamente diferite (există diverse îmbunătățiri sau modificări aduse executabilelor care pot, de asemenea, să influențeze execuția job-urilor).
- modelarea comportamentului job-ului folosind un număr mare de parametrii generează un număr foarte mare de subseturi de date cu cardinalitate mică ceea ce poate să genereze estimatori insuficient de robuști.

În cadrul tezei și a rezultatelor prezentate în continuare, se consideră două chei folosite pentru delimitarea job-urilor. Prima cheie este formată din numele producției, numele site-ului, modelul de procesor și configurația de hyperthreading. A doua cheie este formată din numele producției, numele site-ului și numele nodului de execuție. Nodul de execuție conține o combinație de procesor și configurație de hyperthreading.

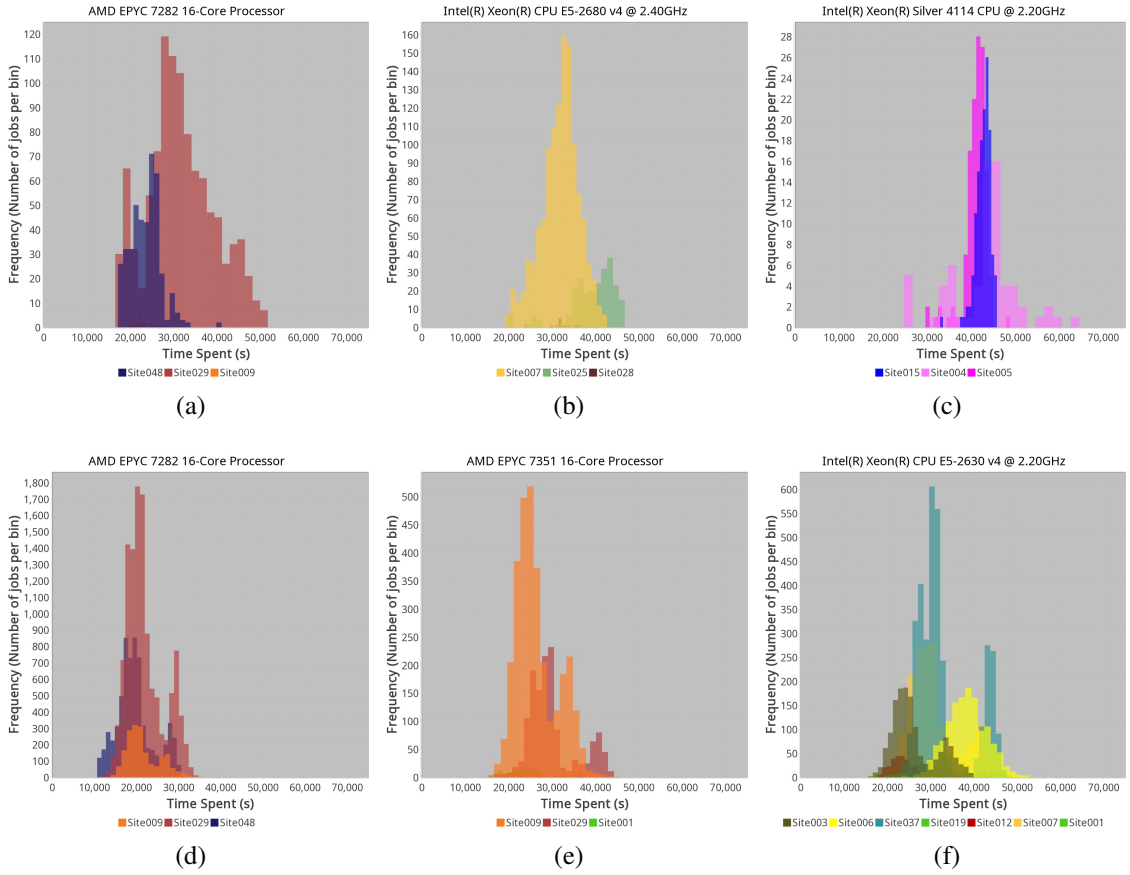



Figura 3.5: Distribuțiile duratei de execuție a job-urilor împărțite după numele site-ului și grupate după modelul de procesor și configurația de hyperthreading. Histogramele 3.5a, 3.5b și 3.5c sunt pentru producția LHC19_cpubench_pp, iar 3.5d, 3.5e și 3.5f sunt pentru producția LHC22e1_extra. Pentru toate modelele prezentate, hyperthreading-ul este activat.

## 3.2 Estimarea duratei job-urilor de tip MonteCarlo

Pe baza rezultatelor prezentate în secțiunea 3.1, considerăm că numele producției, site-ul, modelul de procesor și configurația de hyperthreading ca parametrii principali pe care îi folosim pentru estimarea timpului de execuție a unui job. Pe parcursul acestei teze, se consideră cheie o serie de termeni folosiți pentru împărțirea unui set de date în subseturi. Pe baza histogramelor prezentate în secțiunea 3.1, se observă că pentru un număr suficient de mare de intrări în subseturi, acestea au o distribuție normală asupra cărora putem aplica proprietăți ale distribuției normale.

Alterarea timpului maxim de execuție a unui job (TTL-ul) trebuie realizată sub niște cerințe clar formulate. Astfel, în această teză, sunt definite următoarele cerințe:

- **R1**: algoritmul trebuie să estimeze valoarea maximă a timpului de execuție a unui job astfel încât să prevină terminarea prematură a job-ului: un TTL prea mic (ex. media sau mediana) poate omorî job-urile prea devreme.
- **R2**: algoritmul trebuie să funcționeze în cazul în care distribuțiile sunt prea largi sau greu modelabile. Majoritatea subseturilor de date observate au distribuție normală, însă sunt

cazuri care prezintă anomalii. Algoritmul trebuie să funcționeze și în acest caz.

- **R3**: algoritmul trebuie să fie rapid deoarece zeci de job-uri sunt planificate în fiecare secundă în Grid. Un algoritm prea lent aduce penalități de performanță.
- **R4**: algoritmul trebuie să folosească un număr redus de resurse. Deoarece milioane de job-uri rulează zilnic în Grid, salvarea informațiilor despre acestea pe o perioadă lungă de timp nu este fezabilă.
- **R5**: algoritmul trebuie să păstreze corectitudinea atunci când este transpus în linii de cod (să nu fie afectat de erori de calcul cauzate de lucru cu valori mari)
- **R6**: algoritmul trebuie să se adapteze la dinamica Grid-ului.

Fie $D_k$ definiția formală a unui subset de date (Ecuația 3.1) în care fiecare $x_i \in D_k$ reprezintă timpul de execuție a unui job de simulare Monte Carlo, terminat cu starea `DONE`, pentru care job-ul are cheia $k$: pentru o cheie dată $k$ (ex., "Producția P/SiteXYZ/CPU Model T/hyperthreading este activat"), se generează un subset de date cu valorile timpilor de execuție a job-urilor Monte Carlo care au terminat cu succes și ale căror proprietăți se regăsesc în cheia $k$ (ex., job-ul aparține Producției P, a rulat pe un nod de execuție de pe SiteXYZ care are CPU Model T ca procesor și pentru care hyperthreading-ul este activat).

$$D_k = \{timeSpent(job) \mid job \in MCJobs, status(job) = DONE, key(job) = k\} \tag{3.1}$$

Fie $\sigma$ deviația standard a subsetului de date și $\bar{x}$ media aritmetică a valorilor din subsetul de date. Probabilitatea ca o valoare din subsetul de date să fie la maxim două deviații standard de medie este de 95%

Considerând proprietățile unei distribuții normale și definiția formală a subsetului de date (Ecuația 3.1), formula propusă pentru estimarea unui TTL maxim este prezentată în Ecuația 3.1. Ecuația propune ca TTL-ul estimat să fie la două deviații standard față de valoarea maximă a setului de date. Acest model permite ca job-urile să nu fie omorâte prea devreme și ca ecuația să funcționeze și pe distribuții bi-modale sau oblice.

$$eTTL_k = max(D_k) + 2 * \sigma \tag{3.2}$$

Formula propusă prin Ecuația 3.2 funcționează în cazul în care există suficient de multe job-uri în setul de date($|D_k| = n$). Dacă $n$ este prea mic, atunci TTL-ul estimat poate fi prea mic și poate omorî prematur job-urile. Astfel, este nevoie ca TTL-ul estimat să fie ponderat în funcție de cardinalitatea mulțimii. Cu cât $n$ crește, cu atât crește ponderea pentru estimatorul propus. În cadrul acestei teze, se propune ponderarea TTL-ului estimat cu TTL-ul cerut inițial de către job (notat în continuare $reqTTL$). Formula de ponderare este prezentată în Ecuația 3.3.

$$\begin{aligned} wTTL_k &= w_0 * reqTTL_k + w_1 * eTTL_k \\ w_0 &= \frac{f_0(n)}{f_0(n)+f_1(n)} \\ w_1 &= \frac{f_1(n)}{f_0(n)+f_1(n)}, \text{where } |D_k| = n \end{aligned} \tag{3.3}$$

Pentru simplitate, în cadrul testelor efectuate în teză, au fost alese pondrile din Ecuația 3.4. Aceste permit ca după 50 de job-uri, TTL-ul modificat (ponderat) să fie media dintre TTL-ul original și TTL-ul estimat. După 1000 de job-uri, TTL-ul estimat va compune 95% din TTL-ul modificat.

$$f_0(n) = 50, f_1(n) = n, n \in \mathbf{N}, n > 0 \tag{3.4}$$

Ecuațiile 3.5 și 3.6 prezintă formula generală de ponderare atunci când mai multe chei sunt luate în considerare.

$$D_{i,k} = \{timeSpent(job) \mid job \in MCJobs, status(job) = DONE, key_i(job) = k\}, i = 1..m. \tag{3.5}$$

$$\begin{aligned} wTTL_{key_{1..m}} &= w_0 * reqTTL + \sum_{i=1}^{m} w_i * eTTL_{i,k} \\ w_0 &= \frac{f_0(\sum_{i=1}^{m} |D_{i,k}|)}{f_0(\sum_{i=1}^{m} |D_{i,k}|) + \sum_{i=1}^{m} f_i(|D_{i,k}|)} \\ w_i &= \frac{f_i(|D_{i,k}|)}{f_0(\sum_{i=1}^{m} |D_{i,k}|) + \sum_{i=1}^{m} f_i(|D_{i,k}|)} \\ eTTL_{i,k} &= estimation(D_{i,k}) \end{aligned} \tag{3.6}$$

Pentru implementarea algoritmului propus, trebuie îndeplinite cerințele specificate anterior. **R1** și **R2** sunt îndeplinite prin calcularea TTL-ului estimat conform formulei 3.2 și ponderarea acestuia folosind formula 3.3.

Pentru îndeplinirea **R3**, **R4** și **R5**, implementarea propusă în cadrul acestei teze folosește algoritmul lui Welford pentru calcularea mediei și varianței [61, 62] deoarece acesta presupune calculul incremental (și nu salvarea întregului set de date și recalcularea mediei și deviației standard la fiecare pas). Mai mult, implementarea folosește baze de date și calcule direct în cadrul procesului de inserare sau actualizare a intrărilor pentru acces rapid al datelor. TTL-ul estimat, media și varianța sunt calculate în avans, atunci când un job termină execuția și este adăugat în setul de date. Cu toate acestea, valoarea maximă a setului de date este actualizată doar dacă valoarea nu este considerată a fi outlier.

Pentru **R6**, o aplicație suplimentară trebuie implementată astfel încât, dacă o producție nu ru-

leză pentru o perioadă de timp, atunci metadatele de estimare să fie eliminate.

**Algorithm 1** Modificarea TTL-ului job-ului pe baza estimării propuse

1: $k = key(j)$;
2: **if** $isMonteCarlo(j)$ and $existsEstimationMetadata(k)$ **then**
3: $eTTL = getETTL(k)$;
4: $n = getNum(k)$;
5: $reqTTL = getReqTTL(j)$;
6: $wTTL = getWeightedTTL(eTTL, reqTTL, n)$;
7: $setNewTTL(j, wTTL)$;
8: **end if**

**Algorithm 2** Actualizarea metadatelor atunci când un job termină execuția cu succes

1: **if** $isMonteCarlo(j)$ and $jobStatus(j) == DONE$ **then**
2: $k = key(j)$;
3: $timeSpent = timeSpent(j)$;
4: $updateETTL(k, timeSpent)$;
5: **end if**

Algoritmii 1 și 2 prezintă pseudocodul folosit pentru implementarea soluției propuse.

## 3.3 Îmbunătățiri aduse planificării job-urilor de simulare Monte Carlo în Gridul ALICE

Această secțiune prezintă rezultatele validării algoritmului propus în Secțiunea 3.2.

Pentru testare, a fost implementată o aplicație care consideră job-urile în ordinea în care au fost executate în Grid și le parcurge pe rând, atribuindu-le un TTL modificat. La final, se verifică dacă TTL-ul modificat ar fi omorât sau nu un job. Scopul algoritmului propus este să reducă TTL-ul cerut de un job.

Rezultatele din această secțiune au fost realizate pe seturile date din Octombrie 2022 pentru `LHC19_cpubench_pp` și Octombrie-Noiembrie 2022 pentru `LHC22e1_extra`. În această secțiune sunt prezentate, cu diferite culori, următoarele: cu roșu este reprezentant timpul real de execuție a unui job; cu negru este reprezentat rezultatul estimării atunci când cheia este compusă din <Producție, Site, CPU, hyperthreading>; cu albastru este reprezentat rezultatul estimării atunci când cheia este compusă din <Producție, Site, Host>. În cazul ideal, toate punctele roșii ar trebui să fie sub punctele de estimare. Însă, seturile de date prezintă outliere, astfel încât scopul este ca TTL-ul să fie redus, iar numărul de job-uri care ar fi fost omorâte să fie foarte mic.

Figurile 3.6 și 3.7 arată rezultate obținute pentru `LHC19_cpubench_pp` și, respectiv `LHC22e1_extra` atunci când cheia folosită este formată din numele producție, site-ului, modelul de procesor și configurația de hyperthreading.

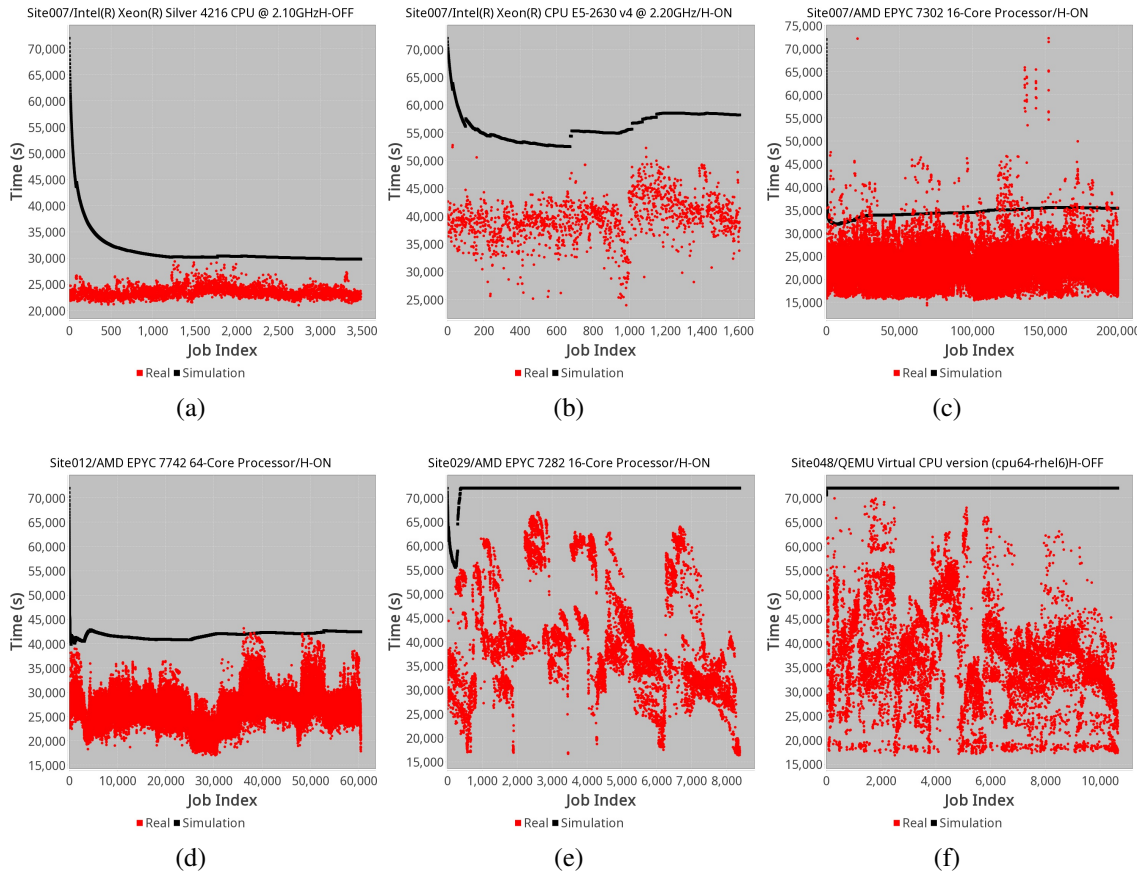



Figura 3.6: Rezultatele obținute pentru LHC19_cpubench_pp. Setul de date folosit este din Octombrie 2022 (Tabela 3.1)

Figurile 3.6a, 3.6b, 3.6c și 3.6d arată cum TTL-ul modificat scade de la 72000s (TTL-ul cerut de job) la o valoare mai mică, iar Figures 3.6e și 3.6f arată cum se comportă algoritmul pe seturi de date greu predictibile (distribuții foarte largi) - în acest caz, se păstrează TTL-ul original. Trebuie menționat că în Figura 3.6f este reprezentat un procesor virtualizat generic (`QEMU Virtual CPU`), astfel încât mai multe modele de procesor pot rula aceste job-uri. Deși în Figura 3.6d apar și job-uri care ar fi fost omorâte de simulare, trebuie menționat că acestea reprezintă doar 2‰din job-urile acelui set de date.

Figura 3.7a conține aproape 100000 job-uri, cu mai puțin de 1‰job-uri care ar fi fost omorâte prematur. În contrast, Figura 3.7b conține aproape 750 jobs, arătând faptul că algoritmul funcționează și pentru seturi de date mici. Figurile 3.7b și 3.7c arată că algoritmul funcționează și pe producții bi-modale. În cazul curent, TTL-ul este redus de la 72000s la aproape 36000s și, respectiv, 44000s.

Figurile 3.6 și 3.7 arată că folosind cheia formată din <Producție, Site, Model de procesor, hyperthreading>, TTL-ul este redus cu până la 50%, în unele cazuri și mai mult. Acest lucru înseamnă că se poate dubla numărul de job-uri planificabile într-un slot de Grid.

Figurile 3.8 și 3.9 prezintă rezultatele obținute atunci când cheia de împărțire a datelor este formată din producție, site și nodul de execuție (denumit în continuare host). Folosind această


(a) (b) (c)

Figura 3.7: Rezultatele obținute pentru `LHC22e1_extra`. Setul de date utilizat este din Octombrie-Noiembrie 2022 (Tabela 3.1)

cheie, peste 18000 chei unice au fost create pentru fiecare producție.

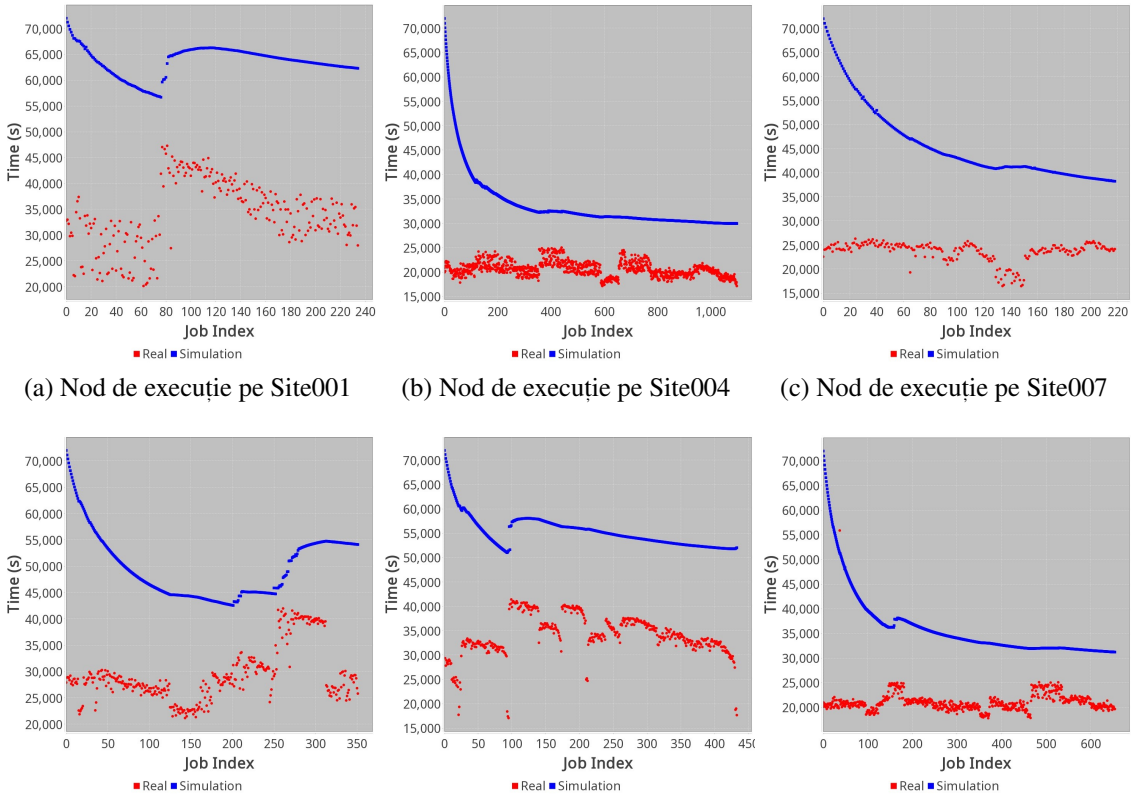
Figurile 3.8b, 3.8f și 3.9a arată că TTL-ul propus scade. Figurile 3.8a, 3.8d și 3.9c arată cum algoritmul se comportă la modificările de pe host.

În cadrul testelor efectuate, s-a considerat și combinarea celor două tipuri de estimatori (primul bazat pe <Producție, Site, Model de procesor, hyperthreading>, cel de-al doilea bazat pe <Producție, Site, Host>). Rezultatele arată că algoritmul converge la o valoare stabilă cel mai rapid atunci când cheia este bazată pe <Producție, Site, Model de procesor și hyperthreading>, însă, folosind cea de-a doua cheie se obțin rezultate mai bune pentru site-urile cu comportament greu modelabil. Cu toate acestea, în cadrul testelor realizare, combinarea celor două tipuri de estimări și ponderarea lor cu TTL-ul inițial nu reduce suficient de rapid TTL-ul.

Pentru o testare extensivă a implementării propuse, algoritmul a fost rulat pe un set de date colectat de către echipa Grid-ului ALICE pe durata a 6 luni. Setul de date conține aproximativ 7 milioane de job-uri, împărțite în 180 de producții, dintre care 40 sunt producții Monte Carlo, restul fiind bazate pe executabile intensive I/O. Algoritmul a fost testat pe toate producțiile pentru a studia comportamentul acestuia și în alte cazuri decât cele urmărite în cadrul studiului.

Figura 3.10 arată rezultatele obținute atunci când algoritmul este aplicat producțiilor I/O intensive. Rezultatele sunt bune și în aceste cazuri. Acest lucru se întâmplă datorită configurației dispozitivelor de stocare de pe site-uri: job-urile sunt planificate în localitatea datelor de care au nevoie, iar toate nodurile de execuție au acces uniform la același mediu partajat (local) de stocare. Folosind cheia <Producția, Site-ul, Modelul de procesor, hyperthreading>, se observă cum TTL-ul estimat scade de la 36000s la 2000s, reducând TTL-ul cu aproape 95%. Acest lucru demonstrează că algoritmul prezentat poate fi aplicat și în alte situații, atât timp cât cheia aleasă generează seturi de date distribuite normal.

Figura 3.11 prezintă rezultatele obținute pentru producții Monte Carlo cu date colectate de-a lungul a 6 luni. În acest set de date, diferit de seturile de date folosite de-a lungul analizei, producția `LHC22e1_extra` (aceeași ca cea considerată și în analiză) are aproximativ 4750K




(a) Nod de execuție pe Site001 (b) Nod de execuție pe Site004 (c) Nod de execuție pe Site007

(d) Nod de execuție pe Site012 (e) Nod de execuție pe Site029 (f) Nod de execuție pe Site048

Figura 3.8: Rezultatele obținute pentru `LHC19_cpubench_pp` când cheia este formată din <Producție, Site, Host>. Setul de date folosit este din Octombrie 2022 (Tabelul 3.1)

joburi, iar `LHC22i2b` și `LHC22e1_tr` aproximativ 439K și, respectiv, 231K job-uri.

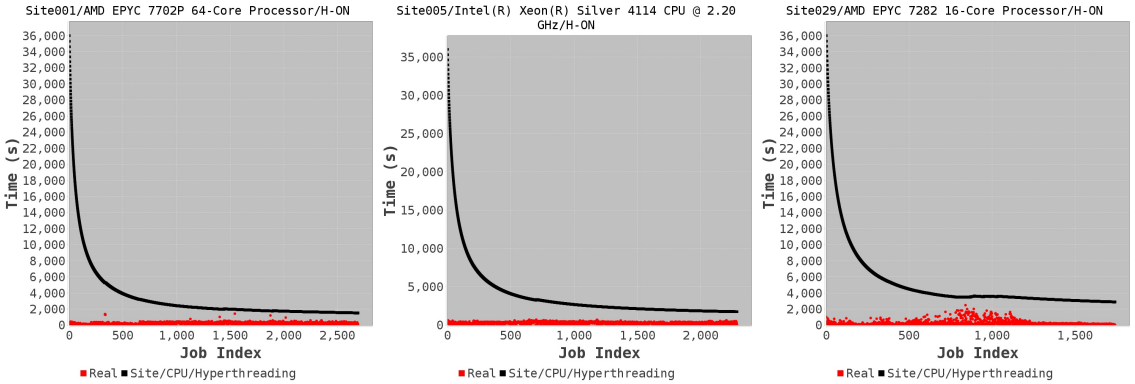
## 3.4 Discuție pe baza rezultatelor

Cele două chei propuse, precum și combinația lor, au dus la rezultate bune, în care TTL-ul inițial a fost scăzut în majoritatea cazurilor. Cheia care converge cel mai rapid către o valoare bună este formată din producție, site, modelul de procesor și configurația de hyperthreading. A doua cheie considerată (formată din numele producție, site-ul și host-ul de execuție) converge lent către o valoare bună.

Pentru seturile de date considerate, s-au obținut aproximativ 300 de chei per producție pentru prima cheie și aproximativ 18000 de chei per producție pentru a doua cheie. Luând în considerare faptul că prima cheie converge mai rapid și cazurile în care cea de-a doua cheie are rezultate mai bune sunt izolate (doar anumite site-uri, precum cele care ascund numele procesorului prin virtualizare), pentru implementarea în Grid se va considera cheia formată din numele producției, numele site-ului, modelul de procesor și configurația de hyperthreading.

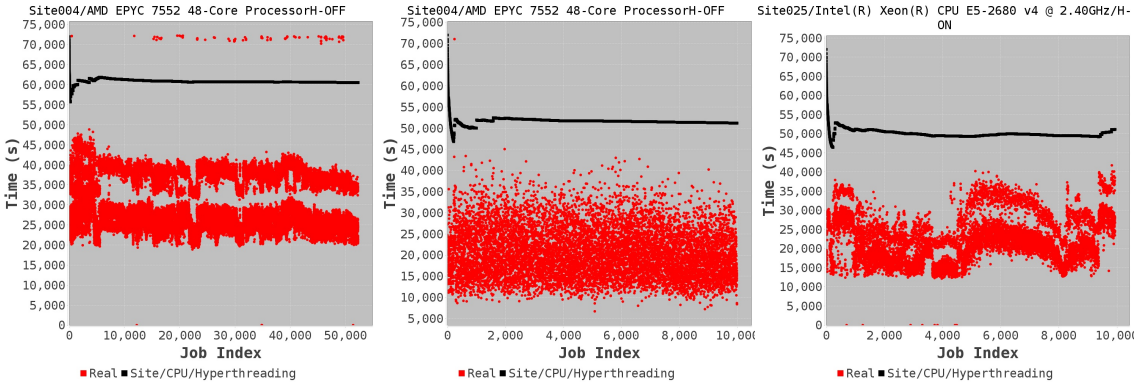

(a) Nod de execuție pe Site004 (b) Nod de execuție pe Site007 (c) Nod de execuție pe Site048

Figura 3.9: Rezultatele obținute pentru `LHC22e1_extra` atunci când cheia este formată din producție, site și host. Setul de date folosit este din Octombrie-Noiembrie 2022 (Tabela 3.1)




(a) QA_LHC22i2b_Stage1 (b) QA_LHC22i2b_Stage1 (c) QA_LHC22e1_tr_Stage1

Figura 3.10: Rezultate obținute pentru QA_LHC22i2b_Stage1 and QA_LHC22e1_tr_Stage1. Aceste producții sunt I/O intensive.




(a) LHC22e1_extra (b) LHC22i2b (c) LHC22e1_tr

Figura 3.11: Rezultate obținute pentru LHC22e1_extra, LHC22i2b și LHC22e1_tr. Aceste producții sunt CPU intensive.

# Capitolul 4

# Îmbunătățiri aduse în infrastructura de calcul pentru activități de cercetare și învățare

Activitățile de cercetare și învățare se bazează pe infrastructura de calcul. Fie că este vorba de resurse eterogene pentru operații complexe [4, 10, 11, 14, 18, 26, 27] ori pentru facilitarea operațiilor de învățare prin intermediul platformelor e-learning [4, 5, 25, 28, 29, 30, 31, 32, 33, 34, 35, 38, 63], administratorii de sistem au ca activitate principală îmbunătățirea mediului de lucru. Pe baza unei infrastructuri corespunzătoare, procesele de educație, cercetare și dezvoltare se pot desfășura mai ușor.

Acest capitol prezintă mai multe îmbunătățiri implementate pentru dezvoltarea mediului de educație și cercetare. Aplicarea acestor contribuții s-a realizat în cadrul Univesității Politehnica din București. Astfel, a fost realizată o **structură de creare automată a cursurilor** într-o platformă e-learning (Moodle), pe baza planurilor de învățământ. Pe baza contractelor de studii, s-a realizat o **înrolare automată a studenților** la cursuri.

Pentru îmbunătățirea mediului de lucru a studenților și cercetătorilor care dezvoltă proiecte pe baza FreeBSD, s-au aplicat diverse modificări astfel încât FreeBSD să poate fi rulat ca un **nod de calcul în OpenStack** (fără componenta de rețea). Acest capitol prezintă teste realizate în **integrarea FreeBSD ca nod de execuție în cluster**. În cadrul tezei sunt arătate **îmbunătățirile aduse procesului de migrare live a mașinilor virtuale folosind bhyve**, hypervisorul din FreeBSD.

## 4.1 Îmbunătățiri aduse procesului de automatizare a generării structurii de cursuri în platforme e-learning

Moodle este o platformă de e-learning care permite utilizatorilor săi să personalizeze conținutul. Administratorii platformei pot configura platforma după nevoile instructorilor prin instalarea de plugin-uri (din interfața web) sau prin crearea unor operații automate de configurare a infrastructurii.

Moodle este o platformă complexă, ce poate fi folosită atât pentru activitățile tradiționale de distribuire de materiale de studiu, învățare și notare, cât și pentru activități inovatoare: analiza comportamentului studenților [1], rapoarte [2], curriculă personalizată pe baza intereselor studentului [64, 65].

Pentru simplificarea procesului de generare a configurării platformei Moodle, teza propune implementarea unor task-uri planificate prin intermediul platformei RunDeck.

În această secțiune, se prezintă implementarea propusă în UPB, bazată pe servicii de administrare a identității utilizatorilor pre-existente în UPB. Figura 4.1 arată procesul de automatizare implementat în UPB.

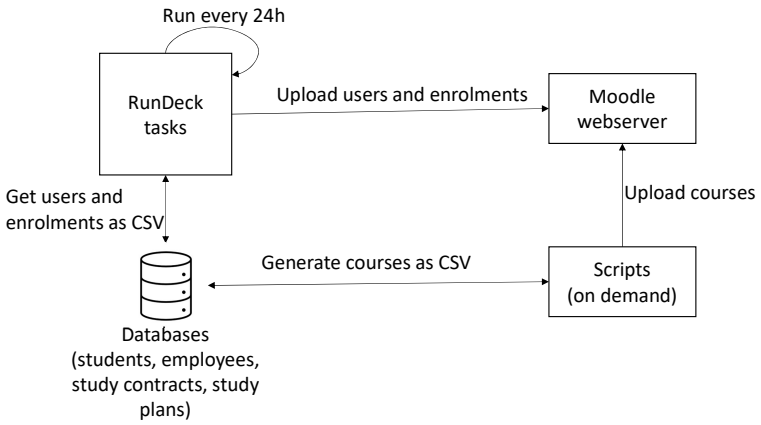



Figura 4.1: Automatizarea generării de cursuri și studenți în Moodle

Deoarece UPB acomodează 15 facultăți și un total de 35000 de utilizatori, iar, anual, se generează aproximativ 11000 de cursuri, este nevoie de o infrastructură dedicată pentru generarea acestora. Astfel, pe baza unor scripturi mySQL , se extrage structura de cursuri din bazele de date interne ale facultăților (informații precum facultatea, numele cursului, ciclul de studii, programul de studii sau specializarea, anul și semestrul). Pe baza acestor informații, precum și a unor scripturi Linux, se generează o previzualizare a cursurilor. În momentul în care responsabilii facultăților acceptă structura din previzualizare, aceasta se generează în platforma de

[1] Moodle Analytics, Online: https://docs.moodle.org/311/en/Analytics, Accesat 12 Octombrie 2023

[2] Moodle Report, Online: https://docs.moodle.org/400/en/Course_overview_report, Accesat 12 Octombrie 2023

e-learning. Deoarece ștergerea de cursuri este computațional intensivă pentru Moodle, s-a ales ca structura de cursuri să fie generată doar la cerere, pentru a evita penalități de performanță în cazul în care trebuie șterse cursuri generate greșit. Fiecare curs este creat pe baza unui template formatat de către administratorii platformei.

De asemenea, înrolarea studenților (aproximativ 30000) la cursuri (fiecare student are 5-7 cursuri în fiecare semestru) nu se poate realiza manual. Tot pe baza unor scripturi mySQL și bash, se extrag informații despre student din contractul de studii și se înrolează la cursurile specifice. Înrolările sunt programate, prin intermediul RunDeck, să ruleze în fiecare noapte.

## 4.2 FreeBSD folosit ca sistem de operare în cluster și Grid

Începând cu 2014[1], multiple îmbunătățiri și proiecte de cercetare au fost dezvoltate în cadrul UPB: procedura de save/restore pentru bhyve [66, 67, 68, 69], proceduri de migrare a mașinilor virtuale folosind bhyve [66, 70], portarea bhyve pe platforma ARM[71, 72, 73, 74], îmbunătățiri aduse suportului "libvdsk" în bhyve, caching la nivel de instrucțiuni[75].

Cu toate acestea, dezvoltarea de funcționalități în cadrul unui sistem de operare are următoarele neajunsuri:

- compilarea codului durează mult timp [76], având în vedere că sistemele de operare au milioane de linii de cod, iar compilarea completă necesită multe resurse computaționale.
- erorile de programare introduse în kernel pot să genereze kernel panic, iar remedierea acestora este anevoioasă, putând duce la pierderea mediului de lucru. De aceea, este nevoie de copii ale datelor (backup-uri).
- codul trebuie testat folosind resurse variate (e.g., mașini virtuale migrate între două noduri identice).
- procesul de recenzie a codului durează mult [77], iar funcționalitățile care au dimensiuni mari trebuie sparte în mai multe părți. Este necesară păstrarea infrastructurii de test, chiar dacă proiectul avansează.
- actualizarea codului cu modificările publicate oficial trebuie realizată periodic.

Având în vedere dificultățile studenților și cercetătorilor în procesul de dezvoltare a modificărilor în cadrul sistemului de operare, este nevoie de existența unui mediu de lucru eterogen care să cuprindă și sistemul de operare FreeBSD.

FreeBSD are numeroase avantaje și este utilizat, în special ca sistem de operare pentru servere, de către companii pentru vitezele mari de transfer de date (Netflix [78], ScaleEngine[2], WhatsApp [79]), ca sistem de operare de bază pentru produse derivate (AsyncOS utilizat de filtrul de

[1]The FreeBSD Foundation's Wiki Page, Online: https://wiki.freebsd.org/SummerOfCode2014/InstructionCachingInBHyVe, Accesat: 2 Noiembrie 2023

[2]ScaleEngine, Online: https://www.scaleengine.com/, Accesat 21 Septembrie 2023

spam IronPort [1], JunOS utilizat de ruterele Juniper [2]), ca soluții de firewall (pfSense [80]), ca soluții de stocare peste rețea (TrueNas[3]). Cu toate acestea, bhyve, hypervisorul de pe FreeBSD nu are o funcționalitate de migrare a mașinilor virtuale adăugată în codul de bază. Procedurile de implementare a acestei funcționalități s-au desfășurat în cadrul Universității Politehnica din București.

FreeBSD este unul dintre cele mai utilizate distribuții BSD. Acest sistem de operare are multiple avantaje, precum existența unui software stabil [81, 82, 83, 84], multiple soluții de securitate (Capsicum [81, 85], folosirea hypervisorului bhyve, care, la rândul lui, folosește Capsicum [81], jails [86] ca soluție de containerizare).

În această secțiune, se prezintă FreeBSD ca sistem de operare folosit pentru dezvoltare și utilizare în medii cluster și cloud, precum și îmbunătățirile aduse procedurii de migrare live a mașinilor virtuale folosind bhyve. La momentul scrierii tezei, FreeBSD nu are suport adăugat pentru rularea ca imagine cloud în OpenStack, nici ca nod de calcul în infrastructura OpenStack. Capitolul prezintă, de asemenea, îmbunătățirile aduse în această direcție.

### 4.2.1 Arhitectura propusă pentru folosirea FreeBSD în infrastructură cluster și cloud

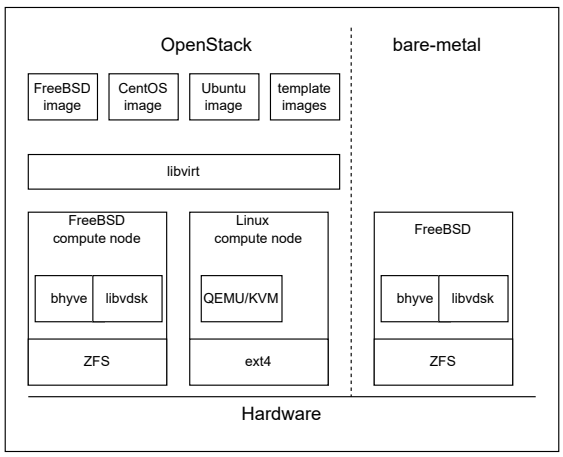



Figura 4.2: Integrarea FreeBSD într-un cluster eterogen

Figura 4.2 prezintă o propunere de integrare a sistemului de operare FreeBSD într-un mediu eterogen. FreeBSD poate fi integrat ca nod de calcul în cluster (rulare bare-metal). Pentru integritatea datelor și backup-uri rapide ale datelor, se poate folosi ZFS ca sistem de fișiere. Mai mult, bhyve poate fi folosit pentru virtualizare. Cu toate acestea, pentru asigurarea unei încărcări balansate între mai multe noduri care folosesc FreeBSD, este necesară procedura de migrare a mașinilor virtuale. Mai mult, pentru folosirea FreeBSD în cloud, acesta se poate integra atât

[1]Cisco, Online: https://www.cisco.com/c/en/us/td/docs/security/wsa/wsa-15-0/release-notes/release-notes-for-wsa-15-0.html, Accesat 21 Septembrie 2023

[2]Juniper, Online: https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/junos-os-overview.html, Accesat 21 Septembrie 2023

[3]iXSystems, Online: https://www.truenas.com/, Accesat 21 Septembrie 2023

ca imagine disponibilă utilizatorului în OpenStack (alături de celelalte imagini existente), dar și ca nod de calcul (pentru administrarea mașinilor virtuale). Prin intermediul libvirt, se poate abstractiza accesul la nodul de execuție, putând avea în paralel atât noduri de Linux, cât și noduri de FreeBSD.

Infrastructura prezentată în Figura 4.2 a fost propusă spre implementare în structura cluster și cloud a UPB. Cu toate acestea, la momentul realizării infrastructurii, FreeBSD nu avea suport pentru funcționarea ca nod în OpenStack, fiind necesare anumite modificări, iar bhyve nu avea integrată funcționalitatea de migrare a mașinilor virtuale. Implementarea acestei infrastructuri în UPB este necesară cercetătorilor și studenților care lucrează în cercetare și dezvoltare în cadrul sau folosind FreeBSD[1].

Pentru adăugarea FreeBSD ca nod de execuție în cadrul arhitecturii, au fost folosite două servere cu Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz (14 core-uri, hyperthreading activat) și cu 380GB of RAM. Aceste două servere sunt noduri similare cu cele folosite în instanța de OpenStack a UPB care le virtualizează. Deoarece bhyve nu are implementată funcționalitatea de virtualizare nested (pornirea unei mașini virtuale din cadrul unei mașini virtuale deschisă cu bhyve), la momentul realizării testelor, unul dintre servere a folosit "FreeBSD 13.0 amd64", celălalt rulând diverse distribuții de Linux și o mașină virtuală de FreeBSD folosind hypervisorul kvm. În Tabelul 4.1 sunt sumarizate rezultatele testării. FreeBSD rulat direct pe server nu generează probleme, rularea bhyve în KVM, pe acest model de server, nu funcționează. Multiple configurații ale sistemului gazdă și ale hypervisorului KVM au fost încercate (dezactivarea `x2lapic`, folosirea `host-model` sau `host-passthrough` în libvirt etc.).

Tabela 4.1: FreeBSD și bhyve bare-metal și virtualizat

| | **FreeBSD bare-metal** | **CentOS 8 (host)** | | **Ubuntu (host)** |
|---|---|---|---|---|
| | | **FreeBSD12 (guest)** | **FreeBSD 13 (guest)** | **FreeBSD13 (guest)** |
| **bhyveload** | ok | VM is blocked (no interraction) | kernel panic in bhyve guest | kernel panic in bhyve guest |
| **uefi** | ok | VM is blocked (no interraction) | VM is blocked (no interraction) | VM is blocked (no interraction) |

La momentul realizării testelor, suportul pentru `cloud-init` (necesar pornirii unei mașini virtuale în OpenStack și realizării automate a configurațiilor de pornire a mașinii) era încă în dezvoltare[2]. Astfel, FreeBSD a fost testat doar ca imagine `qcow2` privată în OpenStack, iar autentificarea s-a făcut cu nume de utilizator și parolă. Deși FreeBSD funcționează corespunzător, bhyve nu a putut fi testat din cauza erorilor afișate în Tabelul 4.1.

[1] Proiecte FreeBSD în UPB, Online: https://github.com/FreeBSD-UPB/freebsd-src/wiki, Accesat 6 Septembrie 2023

[2] The FreeBSD Foundation, Online: https://freebsdfoundation.org/project/freebsd-as-a-tier-i-cloud-init-platform/, Accesat 6 Septembrie 2023

Pentru integrarea FreeBSD ca nod de execuție în OpenStack, sunt necesare modificări în cadrul infrastructurii OpenStack. Pe lângă modificările necesare adăugate în libvirt și serviciile OpenStack Nova și Neutron pentru recunoașterea FreeBSD ca sistem de operare, este nevoie de dezactivarea serviciului OpenStack `oslo-privsep`. Acesta din urmă folosește capabilitățile existente în Linux pentru asigurarea securității și izolării, însă aceste caracterisitici nu există în FreeBSD. Rezultatele obținute sunt deschiderea unei mașini virtuale folosind bhyve, din cadrul OpenStack, însă fără componenta de rețea asociată mașinii virtuale.

### 4.2.2 Îmbunătățirea procedurii de migrare live a mașinilor virtuale folosind bhyve

Migrarea mașinilor virtuale este un mecanism important în administrarea unui cluster, folosit, de obicei, pentru asigurarea unei încărcări egale a nodurilor, dar și pentru golirea unui nod de execuție. Migrarea unei mașini virtuale înseamnă mutarea acesteia de pe un nod pe altul, cu un timp de întrerupere a rulării acesteia cât mai mic. Această secțiune prezintă îmbunătățirile aduse procedeului de migrare live a mașinilor virtuale folosind bhyve.

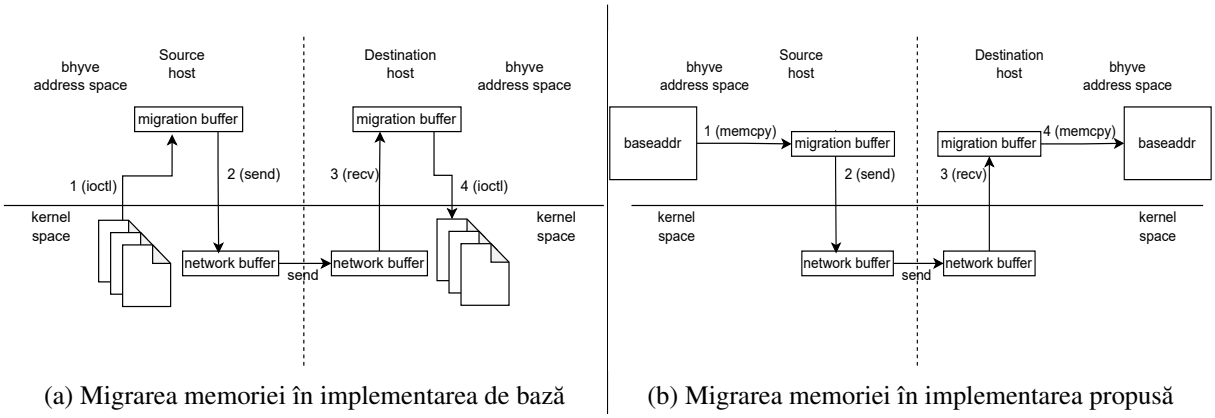
Migrarea live a unei mașini virtuale înseamnă mutarea acesteia pe alt nod, în timp ce aceasta încă functionează.

Procedura de migrare live a mașinilor virtuale folosind bhyve a fost implementată în cadrul unui proiect în UPB[66]. Cu toate acestea, proiectul nu a fost integrat în codul de bază al bhyve și funcționează doar pentru memoria de tip wired (memoria este alocată dinainte și nu poate fi evacuată în swap). Implementarea de bază folosește un mecanism de mutare a datelor mașinii virtuale în runde: în prima rundă se migrează toată memoria virtuală a mașinii virtuale. În rundele următoare, se migrează doar paginile de memorie care au fost modificate între timp. În ultima rundă, se oprește execuția mașinii virtuale pe stația sursă, se migrează paginile rămase și starea dispozitivelor și se pornește mașina la destinație. Paginile modificate sunt identificate prin intermediul unui bit adăugat fiecărei pagini virtuale. În această implementare[66], am descoperit o serie de limitări pentru care această secțiune propune următoarele soluții: adăugarea suportului pentru mașini virtuale cu memoria non-wired și reducerea numărului de schimbări de context în transferul paginilor de memorie.

Pentru adăugarea suportului pentru mașini virtuale cu memoria non-wired, în procesul de verificare a paginilor care trebuie copiate se verifică și stadiul paginii. Paginile alocate în memoria principală sunt tratate ca în implementarea de bază: dacă bitul folosit pentru identificarea modificărilor este setat, pagina trebuie migrată. Paginile care nu au fost alocate sunt ignorate, iar paginile evacuate în zona de swap sunt aduse în memoria principală și se verifică bitul de modificare.

Figura 4.3a prezintă procedeul de migrare a memoriei în implementarea de bază[66] unde un apel de tip `ioctl` este folosit pentru a copia paginile din kernel space în user space. Apoi, acestea sunt transmise din nou, folosind un socket, către destinație, ceea ce înseamnă trecerea

înapoi în kernel space. Figura 4.3b prezintă îmbunătățirile propuse pentru acest proces: în loc să se copieze paginile din kernel space, se folosește zona de memorie a mașinii virtuale mapată în utilitarul din userspace. Pornind de la această adresă, pe baza unui indice al paginii, se poate accesa doar pagina corespunzătoare procesului de migrare.




(a) Migrarea memoriei în implementarea de bază

(b) Migrarea memoriei în implementarea propusă

Figura 4.3: Reducerea numărului de copieri în procesul de migrare a memoriei

Deși modificările prezentate în Figura 4.3 au redus numărul de schimbări de context, testele au arătat că prima rundă de migrare durează la fel de mult ca runda inițială în care se migrează toată memoria alocată a mașinii virtuale. Acest comportament este determinat de modul dual de accesare al memoriei în bhyve: o dată din procesul care a deschis mașina virtuală, prin intermediul mapării memoriei, și o dată prin intermediul tabelei de pagini folosită direct de către mașina virtuală [87] (se accesează direct memoria, fără intermedierea nucleului sistemului gazdă). Astfel, legăturile dintre paginile virtuale accesate prin `vmctx->baseaddr` și paginile fizice corespunzătoare (accesate deja de către mașina virtuală) se face la primul acces din userspace. Această problemă a fost rezolvată folosind apelul `madvise()` cu argumentul `MADV_WILLNEED` astfel ca paginile să fie mapate imediat deoarece ele sunt deja alocate.

Pentru testarea funcționalităților, s-a folosit infrastructura din Figura 4.4. Au fost folosite două sisteme identice cu Intel(R) Core(TM) i7-4790 @ 3,60GHz (4 core-uri) și 16GB RAM care rulează aceeași versiune de FreeBSD. Cele două sisteme partajează un spațiu comun de stocare (prin NFS) pentru imaginile folosite de mașinile virtuale. Prin intermediul testării, se urmărește funcționalitatea corectă a îmbunătățirilor, dar și faptul că nu se introduc penalizări de timp suplimentare. Testarea folosește două tipuri de teste. Primul test pornește mașina virtuală și migrează mașina (operații I/O intensive, încărcare mică). Al doilea tip de test alocă și modifică în mod continuu memoria mașinii virtuale. În testare, s-a urmărit durata fiecărei runde și, în special, a ultimei runde (timpul în care mașina nu rulează), acesta fiind numit în continuare downtime. Rezultatele fac o comparație cu implementarea de bază peste care s-au adăugat funcționalitățile din acest capitol.

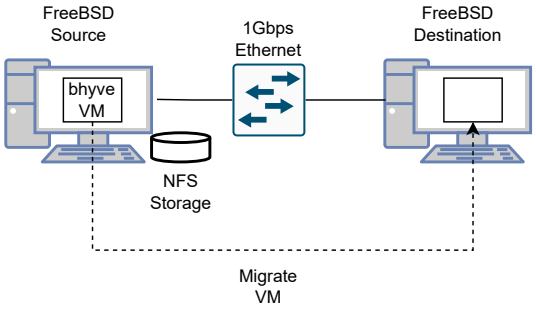
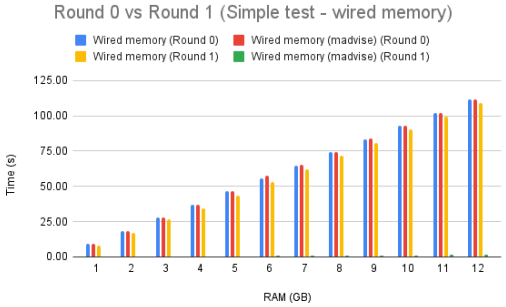
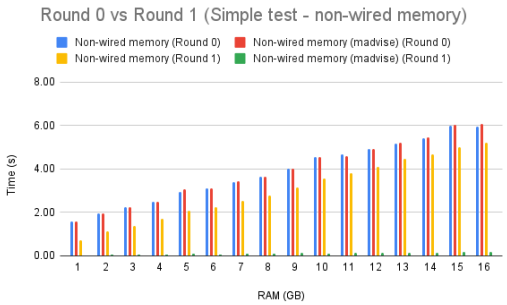



Figura 4.4: Testing setup

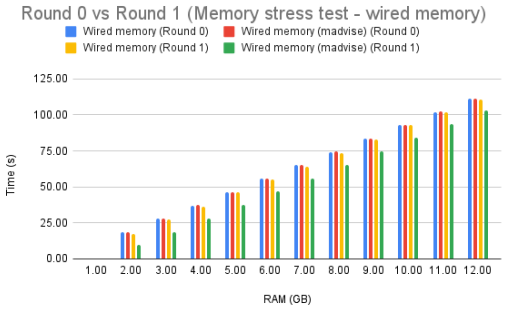
Figura 4.5 prezintă rezultatele testării. În același grafic se prezintă timpii înregistrați pentru runda inițială și prima rundă de migrare. Se observă, de asemenea, rezultatele obținute cu și fără folosirea `madvise`.



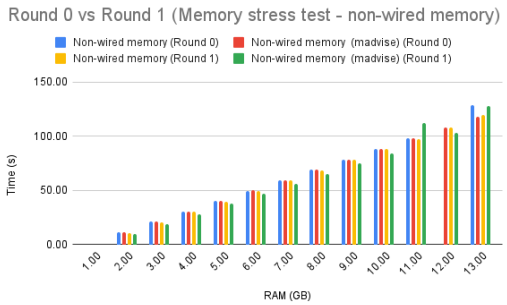

(a) Testul simplu pentru mașină virtuală cu memorie wired




(b) Testul simplu pentru mașină virtuală cu memorie non-wired




(c) Testul de alocare a memoriei pentru mașină virtuală cu memorie wired




(d) Testul de alocare a memoriei pentru mașină virtuală cu memorie non-wired

Figura 4.5: Rundele 0 și 1 înainte și după folosirea `madvise()`

# Capitolul 5

# Securitatea rețelei în arhitecturi cluster și Grid

Asigurarea securității unei infrastructuri complexe este dificilă. De obicei, se folosesc mai multe soluții complementare pentru protejarea rețelei. Cu toate acestea, o problemă a aplicațiilor noi de securitate este că nu sunt testate în medii reale și că modelele de învățare automată pe care le folosesc nu sunt antrenate cu suficiente seturi de date. Acest capitol prezintă o metodologie de testare a unei suite de aplicații într-un mediu real, precum și rezultatele obținute.

O arhitectură tipică de rețea[88, 89] este compusă din mai multe rețele care au funcționalități diferite: rețeaua **backbone**, nucleul care interconectează celelalte subrețele și care conține traficul din toată rețeaua; rețeaua DMZ (demilitarized zone), o zonă accesibilă din Internet; rețeaua **centrului de date** care conține serviciile; și rețeaua utilizatorilor (angajați, studenți, cercetători, vizitatori).

În cadrul testării unei suite de aplicații de securitate (toolkit), există mai mulți pași. În primul rând, se stabilesc tipuri de date folosite pentru analiză și sunt create o serie de scenarii de test care să acopere toate zonele unei rețele. După aceea, se crează o infrastructură pilot în care se testează aceste scenarii. În cadrul acestui capitol, se prezintă aplicarea acestor pași în cadrul proiectului SIMARGL. SIMARGL[2] [90, 91, 92], este o suită de aplicații de securitate (toolkit) [93, 94].

## 5.1 Rețea pilot pentru testarea SIMARGL

Pentru acoperirea cât mai multor scenarii, datele care circulă prin rețeaua pilot trebuie să fie variate: educație și cercetare, tranzacții online, servicii, semnături de documente, trafic enterprise. Astfel, pentru SIMARGL, datele colectate de la fiecare segment de rețea care sunt analizate folosind utilitarele din suită sunt următoarele: pentru backbone, informații de la routere și switch-uri (Netflow, IPFIX, SPAN port); pentru rețeaua DMZ, informații de la soluții

[2]SIMARGL Project, Online: https://simargl.eu/, Accesat 14 Septembrie 2023

de tip firewalls; pentru centrul de date, log-uri de la aplicații, baze de date, DNS și soluții de securitate (IDS, antispam, SIEM, DLP); pentru rețeaua utilizatorilor, log-uri de la soluții IDS sau agenți instalați pe sistemele utilizatorilor pot fi folosite.

Pe baza tipului de trafic și a datelor ce pot fi analizate în cadrul rețelei, s-au generat o serie de scenarii de test. Fiecare scenariu a fost salvat și i s-au atribuit următoarele date: un identificator unic, o versiune, tipurile de atacuri urmărite, segmentul de rețea, aplicațiile sau serviciile implicate sau atacate, datele necesare detectării atacului, o descriere detaliată, informații de anonimizare.

Având aceste informații definite, s-a generat o rețea pilot pentru testarea SIMARGL. Soluțiile centrale au fost instalate în mașini virtuale într-un cluster administrat prin intermediul hypervisorului VMWare vSphere 6 Enterprise Plus. În cadrul acestei rețele pilot s-au integrat diverse soluții din SIMARGL, precum și două rețele monitorizate.

## 5.2 Detecția anomaliilor la nivel de rețea folosind SIMARGL

Această secțiune prezintă două rețele (Figura 5.1) integrate în infrastructura pilot de testare a SIMARGL. Traficul real generat de către cele două rețele (de la utilizatori) a fost analizat folosind două soluții de securitate. Ambele rețele, denumite în cadrul acestui capitol "Rețeaua A" și "Rețeaua B", sunt rețele de cercetare.

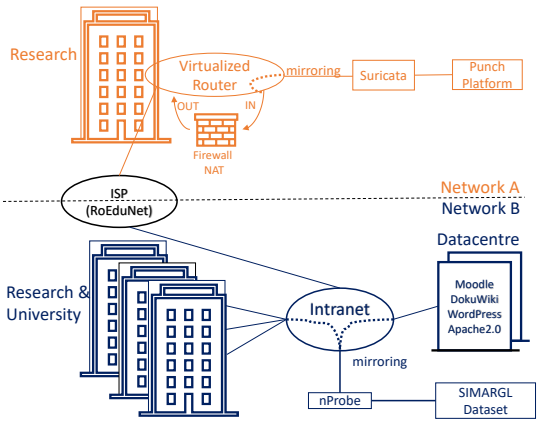



Figura 5.1: Network architectures and mirroring setups

Rețeaua A este o rețea care conține doar trafic de la utilizatori. Rețeaua B este mai complexă și conține trafic de la utilizatorii interni, dar și de la servicii. Diferențele dintre cele două rețele sunt evidențiate în Tabelul 5.1. Cele două rețele diferă din mai multe puncte de vedere: de la complexitate, sursa generatoare de date și direcția conexiunilor, până la serviciile care rulează în cadrul rețelelor.

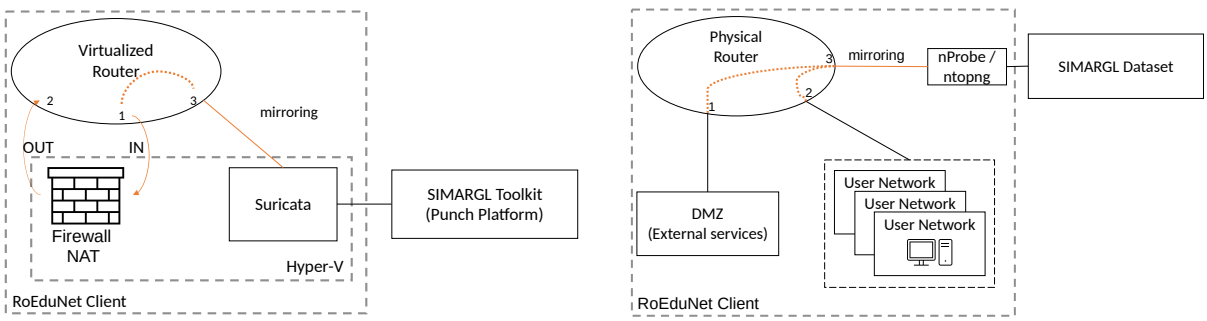
Figura 5.2a arată topologia folosită pentru extragerea de trafic din Rețeaua A care este analizată folosind Punch Platform. Traficul este oglindit (mirror) și parsat de către Suricata, care mai

Tabela 5.1: Comparații între Rețeaua A și Rețeaua B.

| | **Rețeaua A** | **Rețeaua B** |
|---|---|---|
| **complexitate** | o rețea | rețele multiple |
| **router** | router virtual (Hyper-V) | router dedicat |
| **modalitate de oglindire a traficului (mirror)** | folosind Hyper-V | folosind SPAN |
| **sursa de trafic** | cercetători | utilizatori (angajați, studenți, cercetători) și servicii |
| **direcția traficului** | din rețeaua locală în Internet | în ambele direcții |
| **Mecanismul de sondare** | Suricata (VM) | nProbe (nod fizic) |
| **fluctuații ale traficului** | constant | fluctuant |
| **flexibilitate** | rigid (nu poate fi modificat) | flexibil (se pot injecta atacuri controlate) |
| **utilizatori** | $>$ 250 | 200 - 2000 |
| **dispozitive** | $>$ 250 | $>$ 1000 |
| **traffic** | $\sim$500mbps | $\sim$400mbps |
| **servicii expuse în Internet** | - | platforme e-learning (Moodle, learning VMs), Wordpress, Dokuwiki |

departe este analizat folosind Punch Platform (nu au fost adăugate reguli de detecție în Suricata). Arhitectura din Rețeaua A este rigidă și nu permite introducerea unui mediu de rulare complexă de atacuri. Punch Platform, fiind un utilitar deja antrenat, dar nu și testat într-o rețea reală, a analizat pasiv traficul și a raportat anomaliile detectate. În cadrul Rețelei A, au fost detectate scanări verticale (același host, porturi multiple) și orizontale (același port, host-uri multiple), minare de cryptomonede, accesare de site-uri malițioasă (pariuri online, reclame), servere locale de DNS.

Figura 5.2b prezintă topologia folosită pentru Rețeaua B. Traficul a fost extras folosind nProbe și analizat folosind BDE Engine. Arhitectura Rețelei B este una flexibilă, permițând astfel, introducerea unor subrețele malițioase, din cadrul cărora s-au rulat atacuri în mod controlat. Pe baza atacurilor, a rezultat, în primă fază, un set de date folosit în antrenarea modelului de învățare automată a BDE Engine și, în a doua fază, o rețea reală de testare a BDE Engine.




(a) Rețeaua A (b) Rețeaua B

Figura 5.2: Network architectures.

În cadrul Rețelei B au fost introduse două subrețele din interiorul cărora s-au condus atacuri

controlate. Figura 5.3 arată arhitectura folosită, formată din: **Target network**, rețeaua atacată în mod controlat; **Legitimate traffic**, un sector cu trafic de date normal, nemalițios; **External attacker network**, o rețea din care s-au rulat atacuri externe; **Controlled attack environment**, o rețea din cadrul căreia s-au rulat atacuri interne; routerul prin care trece tot traficul din rețea. Orchestrarea atacurilor controlate s-a realizat prin intermediul OpenStack. Traficul rezultat (care trece prin Router), conține atât trafic malițios, cât și trafic normal. Atacurile sunt atât externe (scanare, denial-of-service), cât și interne (scanare, atacuri de tip botnet). Platformele de e-learning atacate nu sunt cele din producție, ci platforme similare asupra cărora s-a generat trafic de utilizator folosind Kubernetes și JMeter.

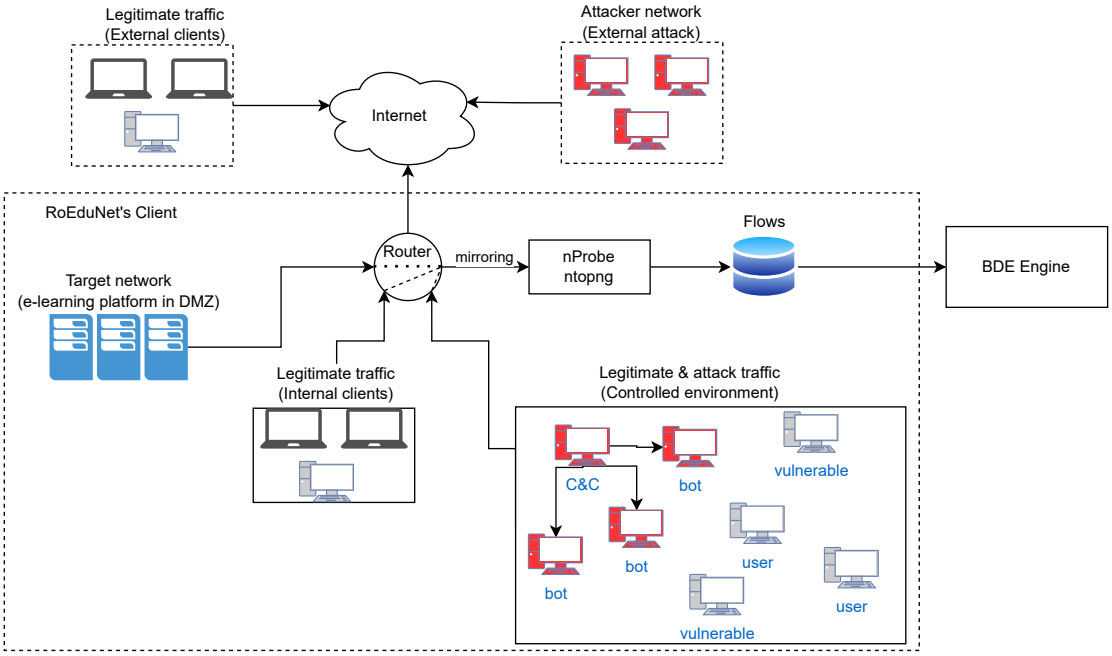



Figura 5.3: Network attack infrastructure

Scanarea rețelei (folosind scanarea porturilor) este primul pas rulat de un atacator pentru a detecta structura rețelei (sisteme de operare, porturi deschise, versiunea aplicațiilor). Astfel, s-au considerat scanări folosind "SYN Scan" (cele mai des folosite și rapide), dar și UDP, FIN, NULL și XMAS Scan. După pasul de scanare, un atacator, de obicei, încearcă să folosească informațiile descoperite pentru continuarea atacului.

Alt tip de atac des folosit este denial-of-service, când se atacă un sistem pentru a-l face inaccesibil. În cadrul acestei teze, au fost considerate atacuri lente (SlowLoris [95] și R-U-Dead-Yet (RUDY) [96]), deoarece acestea simulează clienți înceți. Cererile trimise sunt normale, însă răspunsul în comunicarea cu serverul este întârziat (în cazul RUDY, se trimit mesaje HTTP Post de dimensiune mică, de obicei 1 byte). Deschiderea unui număr mare de astfel de conexiuni lente nu permite serverului să deservească și alți clienți.

Atacurile de tip denial-of-service au eficiență mai mare atunci când sunt distribuite. De obicei, atacurile distribuite sunt lansate din cadrul unor infrastructuri de botnets (sisteme infectate controlate de un server central). Un atac de tip botnet are două părți: în prima parte, malware-ul se multiplică în rețea (fiecare botnet scanează rețeaua și detectează sisteme vulnerabile pe

care le infectează); în a doua parte, un server central de comandă (Command & Control server) anunță boții pe care îi controlează că pot lansa un anumit atac (ex., denial-of-service, minarea cryptomonedelor, etc.). În cadrul SIMARGL, a fost simulat atacul **Mirai** [97]. Acest atac vizează dispozitivele IoT, scanând rețeaua și identificând sistemele cu parole slabe pentru telnet (protul 23). În cadrul arhitecturii, am simulat doar răspândirea malware-ului, nu și atacul denial-of-service.

Pe baza atacurilor, a rezultat un set de date ce poate fi folosit în cadrul etapelor de învățare a algoritmilor de învățare automată. Setul de date conține traficul de rețea în format Netflow, în care adresele IP sunt anonimizate. Pe baza informațiilor interne referitoare la sursele care au generat atacul, traficul din setul de date a fost marcat cu un "LABEL" (câmp identificator) care conține informații despre pachetul respectiv: dacă este trafic normal sau malițios. Pentru traficul malițios, s-a specificat și tipul de atac. Setul de date rezultat a fost folosit pentru antrenarea modelului de învățare automată folosit de BDE Engine.

După faza de antrenare, în cadrul rețelei au fost reluate atacurile, însă fără a specifica tipul traficului (normal sau malițios). Acest pas a fost folosit pentru testarea BDE Engine, care a detectat toate atacurile.

# Capitolul 6

# Concluzii

Administrarea resurselor hardware din cluster și Grid nu este o activitate ușoară. Un administrator de sistem trebuie să ia în calcul mulți factori atunci când stabilește infrastructura, de la performanță la securitate, avându-se în vedere multe cerințe și restricții. În cadrul instituțiilor de cercetare, resursele computaționale (CPU, memorie, disc, rețea), sunt puse la comun și administrate sub forma unei singure entități, un cluster. Deoarece aceste resurse pot să nu fie suficiente pentru rezolvarea problemelor complexe, mai multe clustere se pot interconecta într-o structură distribuită geografic, numită Grid. Dacă resursele trebuie abstractizate și date spre folosință către un utilizator, se obțin platformele cloud.

În cadrul acestei teze este prezentată topologia generală a unui cluster și a unui Grid (Figura 2.1). Pe baza literaturii de specialitate, se identifică nevoile acestor arhitecturi: nevoia de resurse eterogene (software, cât și hardware), nevoia de administrare corectă a resurselor și planificare eficientă a joburilor, nevoia de infrastructuri eficiente energetic, securitate și confidențialitate, facilitate a activităților de învățare și cercetare, varietate a seturilor de date și diversitate a aplicațiilor. De asemenea, teza analizează și două studii de caz: arhitectura cluster și cloud a UPB, precum și arhitectura GRID folosită în cadrul experimentului ALICE (CERN).

Această teză prezintă **îmbunătățiri în cadrul planificăii job-urilor Monte Carlo** prin **determinarea dinamică a timpului maxim de execuție a unui job pe baza istoricului de rulare a job-urilor similare**, cu aplicabilitate în Grid-ul ALICE. Producțiile Monte Carlo au mii de job-uri CPU intensive care sunt distribuite într-un Grid cu eterogenitate mare. Fiecare job din cadrul unei producții pornește din același executabil, cu aceleași variabile, dar cu entropie diferită. Fiecare job are un TTL (Time To Live, timp maxim de execuție) generic. Din cauza eterogenității ridicate a resurselor, TTL-ul curent este setat la o valoare foarte mare, astfel încât să acomodeze toate nodurile de execuție (de la cele mai lente, la cele mai rapide). Cu toate acestea, un TTL mare (aproape 24h) implică planificarea acestuia doar pe slot-uri nou create pe Grid (un slot are durată de viață de 24h). Acest lucru înseamnă că un slot pe un nod de execuție mai rapid și care ar fi putut rula cu succes un job de simulare Monte Carlo într-un timp mai mic, dar care are mai puțin timp de execuție rămas decât TTL-ul maximal cerut, nu poate primi

job-uri noi. Această abordare este suboptimală, deoarece slot-urile de Grid trebuie folosite la capacitate maximă.

În cadrul tezei, este prezentată **analiza a două producții mari de simulare Monte Carlo** (`LHC19_cpubench_pp` și `LHC22e1_extra`) care au rulat în Grid-ul ALICE. În urma analizei, s-au **determinat factorii care influențează timpul de execuției a unui job de tip Monte Carlo**: executabilul (producția), modelul de procesor, configurația de hyperthreading și configurația site-ului. Se analizează, de asemenea, și cazurile particulare (producții bimodale, interval mare de valori ale timpului petrecut de job-uri) și se observă comportamentul producțiilor în puncte diferite de timp.

Pe baza analizei, se determină faptul că folosind și împărțind timpul petrecut de job-uri în Grid pe baza a două tipuri de chei, prima fiind formată din producție, numele site-ului, modelul de procesor, configurația de hyperthreading, iar a doua din producție, numele site-ului și numele host-ului, seturile de date rezultate au distribuție normală (sau cvasi-normală). Astfel, în teză, se propune o **estimare a timpului maxim de rulare a unui job dintr-un set de date** (valoarea maximă a timpului din setul de date, la care se adaugă două deviații standard). Pentru că numărul de job-uri influențează estimarea (seturile mici de date nu pot fi estimate), teza propune și o **ponderare a estimării cu TTL-ul original**: ponderea TTL-ului estimat crește o dată cu cardinalitatea setului de date.

Pentru testare, algoritmul de estimare propus este **implementat și testat pe seturi de date care au rulat deja în Grid-ul ALICE**: job-urile au fost sortate și, fiecărui job, i s-a generat un TTL dinamic (pe baza algoritmului propus). În final, s-a comparat rezultatul obținut cu timpul de execuție al job-ului, scopul fiind ca estimarea să reducă TTL-ul cerut de job, însă să nu îl omoare. Pe baza rezultatelor prezentate în Capitolul 3.3, algoritmul funcționează corect pentru cele două chei propuse. Folosind numele producției, site-ul, modelul de procesor și configuratia de hyperthreading, TTL-ul calculat dinamic converge rapid. Folosind producția, site-ul și host-ul ca și cheie, TTL-ul calculat dinamic scade lent, însă funcționează mai bine pe anumite cazuri particulare. De asemenea, teza prezintă și o testare complexă pe 180 de producții din ALICE, din care doar 40 de simulare Monte Carlo. Rezultatele sunt favorabile și în acest caz. Mai mult, s-a observat că algoritmul funcționează și în cazul producțiilor I/O intensive, demonstrând că alegând o cheie corespunzătoare, algoritmul poate fi aplicat și în alte cazuri.

Această teză prezintă **îmbunătățirile aduse pentru facilitatea activităților de cercetare și învățare**. În cadrul tezei, se descriu operațiile de **administrare automată a cursurilor și înrolărilor la curs** pe baza planurilor de învățământ și a contractelor de studii. Arhitectura propusă este folosită în cadrul platformei e-learning, Moodle, din UPB și generează anual 11000 cursuri folosite de către aproximativ 35000 utilizatori. Arhitectura folosește scripturi bash și MySQL și fluxuri de execuție definite folosind RunDeck.

De asemenea, teza arată necesitatea folosirii FreeBSD în cadrul arhitecturilor cluster și cloud de către studenți și cercetători pentru facilitatea operațiilor de dezvoltare. Teza propune o ar-

hitectură eterogenă, prin care se **integrează FreeBSD atât ca nod de execuție în OpenStack**, cât și ca **nod de execuție în cluster**, cu aplicabilitate în infrastructura UPB. În cadrul tezei se propun **îmbunătățiri ale procesului de migrare live a mașinilor virtuale folosind bhyve**, hypervisorul din FreeBSD. Soluțiile propuse aduc **suport pentru mașini virtuale cu memorie non-wired** (care nu este alocată apriori și care poate fi evacuată în swap) și **reduce numărul de copieri ale datelor și numărul de schimbări de context** prin trimiterea paginilor de memorie virtuală către destinație prin intermediul memoriei mașinii virtuale mapată în userspace.

Teza prezintă modul de **testare a unei suite de utilitare de securitate** (toolkit), cu aplicabilitate în cadrul proiectului SIMARGL. Protejarea securității rețelei este o activitate prioritară, însă soluțiile tradiționale (firewall) nu protejează rețeaua de toate tipurile de atacuri, mai ales cele inițiate din rețeaua internă. De aceea, soluțiile noi de securitate folosesc algoritmi de învățare automată pentru detecția atacurilor. Însă, aceste soluții nu sunt testate folosind date reale și în rețele reale.

În cadrul tezei, se arată cum s-a creat **rețeaua pilot pentru testare** a SIMARGL, cu accent pe două rețele (Rețeaua A și Rețeaua B), monitorizate folosind aplicații din cadrul SIMARGL. Rețeaua A, fiind ridigă, a fost folosită pentru testarea Punch Platform, o soluție de securitate deja antrenată. În cadrul Rețelei A au fost descoperite scanări verticale și orizontale, minare de cryptomonede și accesare de site-uri cu reputație îndoielnică. În cadrul Rețelei B, care este mai flexibilă, au fost **introduse două subrețele** (una extrenă și una internă) din cadrul cărora s-au rulat atacuri în mod controlat. Pe baza atacurilor inițiale (scanare, denial-of-service, atac de tip botnet - Mirai), s-a generat **un set de date public și anonimizat** ce poate fi folosit pentru antrenarea modelelor de învățare automată. Alte etape de atac, nemarcate ca atare, au fost introduse pentru testarea BDE Engine. După etapa de antrenare, aplicația de securitate a detectat cu succes atacurile.

# Bibliografie


[1] S. Singh, Y.-S. Jeong, and J. H. Park, "A survey on cloud computing security: Issues, threats, and solutions," *Journal of Network and Computer Applications*, vol. 75, pp. 200–222, 2016.

[2] R. Rajak, "A comparative study: Taxonomy of high performance computing (hpc)," *Int. J. Electr. Comput. Eng*, vol. 8, no. 5, pp. 3386–3391, 2018.

[3] J. Kolodziej, S. U. Khan, L. Wang, and A. Y. Zomaya, "Energy efficient genetic-based schedulers in computational grids," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 809–829, 2015.

[4] M. Carabas, A. Draghici, G. Lupescu, C.-G. Samoila, and E.-I. Slusanschi, "Integrating parallel computing in the curriculum of the university politehnica of bucharest," in *Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers 24*, pp. 222–234, Springer, 2019.

[5] J. Fabra, S. Hernandez, E. Otero, J. C. Vidal, M. Lama, and P. Alvarez, "Integration of grid, cluster and cloud resources to semantically annotate a large-sized repository of learning objects," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4603–4629, 2015.

[6] S. Campa, M. Danelutto, M. Goli, H. Gonzalez-Velez, A. M. Popescu, and M. Torquati, "Parallel patterns for heterogeneous cpu/gpu architectures: Structured parallelism from cluster to cloud," *Future Generation Computer Systems*, vol. 37, pp. 354–366, 2014.

[7] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 6, pp. 2270–2278, 2016.

[8] M. Sharma and S. Husain, "Analyzing the difference of cluster, grid, utility & cloud computing," *IOSR Journal of Computer Engineering*, vol. 19, no. 1, pp. 55–60, 2017.

[9] T. B. Rehman, "Cloud computing: A juxtapositioning with grid computing," in *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*, pp. 498–503, IEEE, 2017.

[10] A. Larsson, "The need for research infrastructures: a narrative review of large-scale research infrastructures in biobanking," *Biopreservation and Biobanking*, vol. 15, no. 4, pp. 375–383, 2017.

[11] B. K. Daniel, "Big data and data science: A critical review of issues for educational research," *British Journal of Educational Technology*, vol. 50, no. 1, pp. 101–113, 2019.

[12] S. Weisz and M. B. Ferrer, "Adding multi-core support to the alice grid middleware," in *Journal of Physics: Conference Series*, vol. 2438, p. 012009, IOP Publishing, 2023.

[13] M. Martinez Pedreira, C. Grigoras, and V. Yurchenko, "Jalien: the new alice high-performance and high-scalability grid framework. epj web conf 214: 03037," 2019.

[14] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 533–546, 2018.

[15] M. Carabas and P. G. Popescu, "Energy-efficient virtualized clusters," *Future Generation Computer Systems*, vol. 74, pp. 151–157, 2017.

[16] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[17] M. Zakarya and L. Gillam, "Energy efficient computing, clusters, grids and clouds: A taxonomy and survey," *Sustainable Computing: Informatics and Systems*, vol. 14, pp. 13–33, 2017.

[18] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, and J. Kolodziej, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing," *Future Generation Computer Systems*, vol. 51, pp. 61–71, 2015.

[19] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 292–302, 2018.

[20] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[21] T. Vissers, T. S. Somasundaram, L. Pieters, K. Govindarajan, and P. Hellinckx, "Ddos defense system for web services in a cloud environment," *Future Generation Computer Systems*, vol. 37, pp. 37–45, 2014.

[22] A. Carlin, M. Hammoudeh, and O. Aldabbas, "Defence for distributed denial of service attacks in cloud computing," *Procedia computer science*, vol. 73, pp. 490–497, 2015.

[23] P. Xiao, W. Qu, H. Qi, and Z. Li, "Detecting ddos attacks against data center with correlation analysis," *Computer Communications*, vol. 67, pp. 66–74, 2015.

[24] D. Crooks, L. Vlsan, K. Mohammad, M. Carabas, S. McKee, and J. Trinder, "Sissa: Harnessing the power of threat intelligence in grids and clouds: Wlcg soc working group," *PoS*, p. 012, 2018.

[25] J. A. González-Martínez, M. L. Bote-Lorenzo, E. Gómez-Sánchez, and R. Cano-Parra, "Cloud computing and education: A state-of-the-art survey," *Computers & Education*, vol. 80, pp. 132–151, 2015.

[26] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: moving across paradigms," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–39, 2016.

[27] E. C. Dragut, P. Baker, J. Xu, M. I. Sarfraz, E. Bertino, A. Madhkour, R. Agarwal, A. Mahmood, and S. Han, "Cris—computational research infrastructure for science," in *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pp. 301–308, IEEE, 2013.

[28] A. Y. Alsabawy, A. Cater-Steel, and J. Soar, "It infrastructure services as a requirement for e-learning system success," *Computers & Education*, vol. 69, pp. 431–451, 2013.

[29] K. L. Dangwal *et al.*, "Blended learning: An innovative approach.," *Universal Journal of Educational Research*, vol. 5, no. 1, pp. 129–136, 2017.

[30] M. Alier, M. J. Casany, A. Llorens, J. Alcober, and J. d. Prat, "Atenea exams, an ims lti application to solve scalability problems: A study case," *Applied Sciences*, vol. 11, no. 1, p. 80, 2020.

[31] A. Zaini, H. Santoso, and M. Sulistyanto, "Fault tolerance strategy to increase moodle service reliability," in *Journal of Physics: Conference Series*, vol. 1869, p. 012095, IOP Publishing, 2021.

[32] J. Prat, A. Llorens, F. Salvador, M. Alier, and D. Amo, "A methodology to study the university's online teaching activity from virtual platform indicators: The effect of the covid-19 pandemic at universitat politècnica de catalunya," *Sustainability*, vol. 13, no. 9, p. 5177, 2021.

[33] M. Sadikin, R. Yusuf, and A. D. Rifai, "Load balancing clustering on moodle lms to overcome performance issue of e-learning system," *Telkomnika*, vol. 17, no. 1, pp. 131–138, 2019.

[34] J. Prat, A. Llorens, M. Alier, F. Salvador, and D. Amo, "Impact of covid-19 on upc's moodle platform and ice's role," in *Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality*, (New York, NY, USA), p. 765–769, Association for Computing Machinery, 2020.

[35] I. Kureshi, C. Pulley, J. Brennan, V. Holmes, S. Bonner, and Y. James, "Advancing research infrastructure using openstack," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 4, pp. 64–70, 2013.

[36] R. Ferdiana *et al.*, "A systematic literature review of intrusion detection system for network security: Research trends, datasets and methods," in *2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*, pp. 1–6, IEEE, 2020.

[37] M. Aljabri, S. S. Aljameel, R. M. A. Mohammad, S. H. Almotiri, S. Mirza, F. M. Anis, M. Aboulnour, D. M. Alomari, D. H. Alhamed, and H. S. Altamimi, "Intelligent techniques for detecting network attacks: review and research directions," *Sensors*, vol. 21, no. 21, p. 7070, 2021.

[38] A. S. Chow and R. A. Croxton, "Designing a responsive e-learning infrastructure: Systemic change in higher education," *American Journal of Distance Education*, vol. 31, no. 1, pp. 20–42, 2017.

[39] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *Journal of Information Security and Applications*, vol. 29, pp. 27–56, 2016.

[40] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in ddos attacks: trends and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2242–2270, 2015.

[41] P. Bojović, I. Bašičević, S. Ocovaj, and M. Popović, "A practical approach to detection of distributed denial-of-service attacks using a hybrid detection method," *Computers & Electrical Engineering*, vol. 73, pp. 84–96, 2019.

[42] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[43] S. Gupta, A. Singhal, and A. Kapoor, "A literature survey on social engineering attacks: Phishing attack," in *2016 international conference on computing, communication and automation (ICCCA)*, pp. 537–540, IEEE, 2016.

[44] A. Shameli-Sendi, M. Pourzandi, M. Fekih-Ahmed, and M. Cheriet, "Taxonomy of distributed denial of service mitigation approaches for cloud computing," *Journal of Network and Computer Applications*, vol. 58, pp. 165–179, 2015.

[45] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE communications surveys & tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[46] R. R. Karn, P. Kudva, H. Huang, S. Suneja, and I. M. Elfadel, "Cryptomining detection in container clouds using system calls and explainable machine learning," *IEEE transactions on parallel and distributed systems*, vol. 32, no. 3, pp. 674–691, 2020.

[47] M. Russo, N. Šrndić, and P. Laskov, "Detection of illicit cryptomining using network metadata," *EURASIP Journal on Information Security*, vol. 2021, no. 1, pp. 1–20, 2021.

[48] L. R. Bays, R. R. Oliveira, M. P. Barcellos, L. P. Gaspary, and E. R. Mauro Madeira, "Virtual network security: threats, countermeasures, and challenges," *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–19, 2015.

[49] A. Grigoras, C. Grigoras, M. Pedreira, P. Saiz, and S. Schreiner, "Jalien–a new interface between the alien jobs and the central services," in *Journal of Physics: Conference Series*, vol. 523, p. 012010, IOP Publishing, 2014.

[50] S. Schreiner, C. Grigoras, A. Grigoras, L. Betev, and J. Buchmann, "A security architecture for the alice grid services," in *The International Symposium on Grids and Clouds (ISGC)*, vol. 2012, 2012.

[51] M. Bertran Ferrer *et al.*, "Adapting heterogeneous high-performance computing infrastructures for data analysis of the alice experiment at the lhc grid," 2022.

[52] M. M. Pedreira, C. Grigoras, V. Yurchenko, and M. M. Storetvedt, "The security model of the alice next generation grid framework," in *EPJ Web of Conferences*, vol. 214, p. 03042, EDP Sciences, 2019.

[53] M. Storetvedt, L. Betev, H. Helstrup, K. F. Hetland, and B. Kileng, "Running alice grid jobs in containers a new approach to job execution for the next generation alice grid framework," in *EPJ Web of Conferences*, vol. 245, p. 07052, EDP Sciences, 2020.

[54] M. Storetvedt, M. Litmaath, L. Betev, H. Helstrup, K. F. Hetland, and B. Kileng, "Grid services in a box: container management in alice," in *EPJ Web of Conferences*, vol. 214, p. 07018, EDP Sciences, 2019.

[55] M. M. Storetvedt, "A new grid workflow for data analysis within the alice project using containers and modern cloud technologies," 2023.

[56] E. B. Sandvik, "Site sonar-a monitoring tool for alice's grid sites," Master's thesis, The University of Bergen, 2021.

[57] M. Bandieramonte, J. D. Chapman, J. Chiu, H. Gray, and M. Muskinja, "Multi-threaded simulation for atlas: challenges and validation strategy," in *EPJ Web of Conferences*, vol. 245, p. 02001, EDP Sciences, 2020.

[58] J. Scheins, M. Lenz, U. Pietrzyk, N. Shah, and C. Lerche, "High-throughput, accurate monte carlo simulation on cpu hardware for pet applications," *Physics in Medicine & Biology*, vol. 66, no. 18, p. 185001, 2021.

[59] P. Schweitzer, C. Mazel, D. R. Hill, and C. Cârloganu, "Performance analysis with a memory-bound monte carlo simulation on xeon phi," in *2015 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 444–452, IEEE, 2015.

[60] R. A. Tau Leng, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An empirical study of hyper-threading in high performance computing clusters," *Linux HPC Revolution*, vol. 45, 2002.

[61] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[62] A. Chmielowiec, "Algorithm for error-free determination of the variance of all contiguous subsequences and fixed-length contiguous subsequences for a sequence of industrial measurement data," *Computational Statistics*, vol. 36, no. 4, pp. 2813–2840, 2021.

[63] N. Cavus, H. Uzunboylu, and D. Ibrahim, "Assessing the success rate of students using a learning management system together with a collaborative tool in web-based teaching of programming languages," *Journal of educational computing research*, vol. 36, no. 3, pp. 301–321, 2007.

[64] G. Gutu-Robu, D. Mihai, M. Dascalu, M. Carabas, S. Trausan-Matu, S. Choi, K. M. Godfrey, B. A. Brands, and B. Koletzko, "Cohesion network analysis: Customized curriculum management in moodle," in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 175–181, IEEE, 2020.

[65] V. Caputi and A. Garrido, "Student-oriented planning of e-learning contents for moodle," *Journal of Network and Computer Applications*, vol. 53, pp. 115–127, 2015.

[66] M.-E. Mihailescu and M. Carabas, "Freebsd-live migration feature for bhyve," in *AsiaBSDCon*, 2019.

[67] E.-B. Postolache, D. Mihai, M.-E. Mihailescu, S. Weisz, M. Barbulescu, M. Carabas, and N. Tapus, "Suspend feature for multiple devices of same type in bhyve," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, IEEE, 2020.

[68] I. Mihalache, M.-E. Mihăilescu, D. Mihai, M. Carabaş, and N. Ţăpuş, "bhyve-json format and capsicum support for the snapshot feature," in *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 25–29, IEEE, 2021.

[69] I. Mihalache, M.-E. Mihăilescu, D. Mihai, M. Carabas, and N. Ţăpuş, "bhyve-checkpoint functionality based on zfs," in *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 259–262, IEEE, 2022.

[70] M.-E. Mihailescu, D. Mihai, M. Carabas, and N. Tapus, "Improving and testing live migration for bhyve," in *2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, IEEE, 2022.

[71] N.-A. Ivan and M. Carabas, "Finalizing booting requirements for a guest running under bhyvearm,"

[72] A. Elisei and M. Carabas, "bhyvearm64: Generic interrupt controller version 3 virtualization,"

[73] D. Mihai, M.-E. Mihailescu, M. Carabas, and N. Țăpuș, "Booting a linux kernel under bhyve on armv7," in *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 1*, pp. 93–101, Springer, 2021.

[74] A.-C. Martin, D. Mihai, M.-E. Mihailescu, M. Carabas, and N. Tapus, "Symmetric multiprocessor support for bhyve on arm64," in *2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–4, IEEE, 2022.

[75] M. Carabas and C. Carabas, "Instruction caching for bhyve," in *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*, pp. 1–5, 2019.

[76] G. A. Randrianaina, D. E. Khelladi, O. Zendra, and M. Acher, "Towards incremental build of software configurations," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 101–105, 2022.

[77] G. Kudrjavets, J. Thomas, N. Nagappan, and A. Rastogi, "Is kernel code different from non-kernel code? a case study of bsd family operating systems," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 211–222, IEEE, 2022.

[78] R. Stewart and S. Long, "Improving high–bandwidth tls in the freebsd kernel," *FreeBSD Journal*, pp. 8–13, 2016.

[79] R. Reed, "That's billion with a b: Scaling to the next level at whatsapp," *Erlang Factory*, 2014.

[80] K. C. Patel and P. Sharma, "A review paper on pfsense-an open source firewall introducing with different capabilities & customization," *IJARIIE*, vol. 3, pp. 2395–4396, 2017.

[81] J. Anderson, "A comparison of unix sandboxing techniques," *FreeBSD Journal*, 2017.

[82] F.-X. Puig, J. J. Villalobos, I. Rodero, and M. Parashar, "Exploring the potential of freebsd virtualization in containerized environments," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 191–192, 2017.

[83] J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, and J. Sopena, "The battle of the schedulers:{FreeBSD}{ULE} vs. linux {CFS}," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 85–96, 2018.

[84] G. Lencse and S. Répás, "Performance analysis and comparison of different dns64 implementations for linux, openbsd and freebsd," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 877–884, IEEE, 2013.

[85] J. Anderson, S. Godfrey, and R. N. Watson, "Towards oblivious sandboxing with capsicum," *FreeBSD Journal*, 2017.

[86] Y. Takagawa and K. Matsubara, "Yet another container migration on freebsd," *AsiaBSDCon 2019 Proceedings*, pp. 97–102, 2019.

[87] N. Natu and P. Grehan, "Nested paging in bhyve," *The FreeBSD Project, http://people. freebsd. org/neel/bhyve/bhyvenestedpaging. pdf*, 2014.

[88] J. Kugathasan, "Network design report," 12 2017.

[89] J. Ren and T. Li, *Handbook of Technology Management*, ch. Enterprise Security Architecture. John Wiley & Sons, Inc., January 2010.

[90] M. Komisarek, M. Pawlicki, M. Kowalski, A. Marzecki, R. Kozik, and M. Choras, "Network intrusion detection in the wild-the orange use case in the simargl project," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pp. 1–7, 2021.

[91] L. Caviglione, M. Grabowski, K. Gutberlet, A. Marzecki, M. Zuppelli, A. Schaffhauser, and W. Mazurczyk, "Detection of malicious images in production-quality scenarios with the simargl toolkit," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pp. 1–7, 2022.

[92] D. Puchalski, L. Caviglione, R. Kozik, A. Marzecki, S. Krawczyk, and M. Choras, "Stegomalware detection through structural analysis of media files," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pp. 1–6, 2020.

[93] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "Seckit: a model-based security toolkit for the internet of things," *computers & security*, vol. 54, pp. 60–76, 2015.

[94] B. Barak, A. Herzberg, D. Naor, and E. Shai, "The proactive security toolkit and applications," in *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pp. 18–27, 1999.

[95] Y. Tarasov, E. Pakulova, and O. Basov, "Modeling of low-rate ddos-attacks," in *Proceedings of the 12th International Conference on Security of Information and Networks*, SIN '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[96] M. M. Najafabadi, T. M. Khoshgoftaar, A. Napolitano, and C. Wheelus, "Rudy attack: Detection at the network level and its important features," in *The twenty-ninth international flairs conference*, 2016.

[97] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110, 2017.