Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Modelarea sistemelor RAID și măsurarea performanțelor

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea titlului de Inginer
În domeniul Electronică și Telecomunicații
Programul de studii de licență Calculatoare și Tehnologia Informației

2014

Conducător științific: **Conf. Dr. Ing. Ștefan Stăncescu**
Absolvent: **Dragoș – Alexandru Bălăucă**

UNIVERSITATEA "POLITEHNICA" BUCUREŞTI
FACULTATEA DE ELECTRONICĂ TELECOMUNICAŢII SI TEHNOLOGIA INFORMAŢIEI
CATEDRA DE CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI (CTI)

APROBAT Director de Departament
Prof. Dr. Ing. SeverPAŞCA

TEMA PROIECTULUI DE LICENTA A STUDENTULUI

BĂLĂUCĂ I. DRAGOŞ - ALEXANDRU

1. **Titlul temei:**
Modelarea sistemelor RAID şi măsurarea performanţelor

2. **Date iniţiale pentru proiectare:**
Simulator DiskSim 4.0
Mediul de implementare Linux
Limbaj Visual Basic

3. **Conţinutul memoriului:**
Modele de arhitecturi RAID şi prezentarea acestora
Notiuni introductive rebuilding
Moduri de functionare pentru matricile RAID cu disc de rezerva
Experimente pe mediul de simulare DiskSim 4.0 cu parametrii si workload variabile
Simulare, comparaţii, performanţe şi concluzii

4. **Material grafic obligatoriu:**
Rezultate şi grafice comparative

5. **Realizare practică:**
Implementare software

6. **Locul de desfăşurare al lucrării practice**
Catedra de Calculatoare şi Tehnologia Informaţiei

7. **Lucrarea serveşte la:**
Autodotare catedra

8. **Mijloacele materiale puse la dispoziţie de:**
Catedra de Calculatoare şi Tehnologia Informaţiei, UPB

9. **Realizarea practică rămâne în proprietatea:**
Facultatea de Electronică Aplicată şi Tehnologia Informaţiei, UPB

10. **Data eliberarii temei:**
2014

CONDUCĂTOR PROIECT:
Conf. Dr. Ing. Ştefan Stăncescu

Absolvent:
Bălăucă I. Dragoş - Alexandru

Declaraţie de onestitate academică

Prin prezenţa declar că lucrarea cu titlul " *Modelarea sistemelor RAID şi măsurarea performanţelor*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Calculatoare şi Tehnologia Informaţiei*, este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti,

Absolvent Bălăucă Dragoș - Alexandru

_____

(semnătura în original)

DECLARAȚIE DE ORIGINALITATE A PROIECTULUI DE DIPLOMĂ/LUCRĂRII DE DISERTAȚIE[1)]

Subsemnatul BĂLĂUCĂ I. DRAGOȘ ALEXANDRU [2)], posesor al B.I./C.I./pașaport seria RD, nr. 619555, având CNP 1910523440010, am întocmit proiectul de absolvire/diplomă/lucrarea de disertație cu tema:[3)] MODELAREA SISTEMELOR RAIS ȘI MĂSURAREA PERFORMANȚELOR, în vederea susținerii examenului de finalizare a studiilor universitare de licență/masterat, organizat de către Facultatea de ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI, Departamentul CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI din cadrul Universității POLITEHNICA din București, sesiunea IULIE, anul universitar 2014.

Luând în considerare conținutul art. 143 din Legea Educației Naționale nr. 1/2011, al. (4) „*Îndrumătorii proiectelor de diplomă/lucrărilor de disertație răspund în solidar cu autorii acestora de asigurarea originalității conținutului acestora*" și al. (5) „*Este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unui/unei proiect de diplomă/lucrări de disertație*", declar pe proprie răspundere, că acest/această proiect/lucrare este original(ă), fiind rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației în vigoare.

Cunosc faptul că plagiatul sau prezentarea unui/unei proiect/lucrări, elaborat(ă) de alt absolvent sau preluată de pe internet, din manuale și cărți, fără precizarea sursei constituie infracțiune (furt intelectual și nerespectarea dreptului de autor și a proprietății intelectuale) și atrage după sine anularea examenului de diplomă/disertație, precum și răspunderea penală.

Data: 25.06.2014

Semnătura: Bălăucă
(în original)

[1)] Declarația se completează „*de mână*" și reprezintă un document care se depune la înscrierea pentru susținerea examenului de finalizare a studiilor.

Anexa 1

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul : Electronică Aplicată şi Ingineria Informaţiei

Aprobat Director de Departament :

Prof. Dr. Inginer Sever PAŞCA

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Bălăucă I. Dragoş Alexandru, grupa 443A

1. Titlul temei: **Modelarea Sistemelor RAID şi măsurarea performanţelor**

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare): Se vor prezenta structuri de RAID cunoscute şi se va realiza în limbaj C, un simulator de matrici RAID; se vor simula arhitecturi RAID diferite şi se vor compara rezultatele cu datele teoretice; se vor măsura parametri de performanţă în diferite scenarii de funcţionare normală sau cu defecte; se vor simula proceduri de rebuild-ing şi se vor compara timpii de execuţie.

3. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele discipline: Sisteme de Operare, Retele de Calculatoare, Inginerie Software, Arhitecturi de Retele si Internet, Arhitectura Sistemelor de Calcul

4. Realizarea practică/ proiectul rămân în proprietatea: *Universitatea Politehnica Bucuresti*

5. Proprietatea intelectuală asupra proiectului aparţine: *Universitatea Politehnica Bucuresti*

6. Locul de desfăşurare a activităţii: *Universitatea Politehnica Bucuresti*

7. Data eliberării temei: 12.12.2013

CONDUCĂTOR LUCRARE:

Conf. Dr. Ing. Ştefan STĂNCESCU

STUDENT:

Dragoş Alexandru BĂLĂUCĂ

Cuprins

Lista figurilor

NOTĂ INTRODUCTIVĂ

Pentru lucrarea de licență am ales tema "Modelarea sistemelor RAID și măsurarea performanțelor" deoarece în contextul evoluției tehnologiei soluțiile de stocare reprezintă un domeniu de interes foarte important și intens dezbătut.

În zilele noastre, când accesul la informație este absolut necesar la orice oră a zilei, trebuie acordată o atenție deosebită sistemelor de stocare a datelor.

Metodologia lucrării constă în prezentarea teoretică a principalelor tipuri de sisteme profesionale de stocare, caracteristicile de bază ale acestora, avantajele și dezavantajele lor, precum și modul de funcționare al acestora. În continuare se vor alege principalele 4 arhitecturi RAID folosite în viața de zi cu zi și se vor analiza folosind mediul de simulare DiskSim.

În urma rezultatelor obținute în mediu de simulare DiskSim se vor alcătui rapoarte de performanță și grafice comparative între aceste arhitecturi.

Pentru a obține și o privire asupra unei configurații reale aflată în folosință la momentul actual se vor realiza o serie de teste asupra unei matrice RAID 5 și rezultatele vor fi prezentate ca o completare la setul de rezultate obținut pentru simulările realizate cu DiskSim.

Contribuția practică a studentului constă în realizarea mediului de simulare prin recompilarea simulatorului DiskSim și înglobarea acestuia într-o aplicație de sine stătătoare, realizarea unui software de extragere și interpretare a rezultatelor precum și în manipularea cu succes a unor software-uri profesionale de analiză mediilor de stocare.

INTRODUCERE

RAID (Redundant Arrays of Inexpensive Disks sau Redundant Arrays of Independent Disks) este o tehnologie dezvoltată pentru utilizarea simultană a două sau mai multe unităţi HDD într-o configuraţie (matrice) în scopul obţinerii de performanţe crescute alături de o creştere a nivelului de siguranţă a datelor.

În ultimii 25 de ani, standardul RAID s-a schimbat pentru a deveni o caracteristică indispensabila pentru metodele de stocare ale claselor enterprise, dar in acelasi timp şi o varianta de stocare orientata către consumator. RAID imprastie datele pe mai multe discuri dure pentru a creşte toleranţa la erori si pentru a îmbunătăţi performanţa disc-ului.

RAID trebuie privit ca un instrument ce furnizeaza diferite nivele de stocare si care creste performanţa şi disponibilitatea sistemelor clasice. Unitatile de disc continuă să devină mai fiabile si cu capacităţi mai mari. Cu toate acestea, din ce in ce mai multe date sunt stocate, expunandu-le la riscuri atunci când unul dintre discuri se defectează. În plus, faţă de aspectul de disponibilitate a RAID şi tehnologiile de protecţie a datelor, inclusiv oglindire locală şi la distanţă sau de replicare pentru alte sisteme de stocare, RAID oferă, de asemenea, îmbunătăţiri ale performanţei pentru a ajuta la restabilirea balanţei între capacitate şi rezultate excepţionale. [1]

RAID reprezinta un acronim pentru *Redundant Array of Independent Disks* - matrice redundantă de discuri independente si este realizat prin combinarea mai multor discuri într-o unitate logică, unde datele sunt distribuite în unităţi în diferite moduri numite "niveluri RAID".

Tehnologia RAID se imparte în mai multe scheme de stocare, care pot diviza şi reproduce datele între mai multe discuri fizice. Discurile fizice funcţionează ca un singur disc într-o matrice RAID ce este accesată de sistemul de operare. Diferitele scheme sau arhitecturi sunt numerotate cu o cifra dupa cuvântul RAID (spre exemplu: RAID 0, RAID 1 etc.). Fiecare schemă furnizează un raport diferit între două obiective principale: creşterea fiabilităţii datelor şi creşterea performanţei intrare/ieşire. [2]

CAPITOLUL 1

Specificaţii RAID

1.1 Noțiuni generale

Norman Ken Ouchi reușește să obțină în anul 1978 o licenţă ce se intituleză „Sistem pentru recuperarea datelor stocate de pe unităţi de memorie”. Continutullicenței sale descria ceea ce mai târziu urma să fie cunoscut sub termenul de “RAID 5 cu scrieri în fâşii pline”. Această licenţă, din 1978, avea să inoveze prin oglindiriea și/sau duplicitatea discurilor (mai târziu denumită RAID1) precum şi protecţia datelor prin paritate dedicată (ulterior denumită RAID 4).

Cu toate acestea, ideea de RAID a fost introdusă ca atare pentru prima dată în 1987, de către David A. Petterson, Garth A. Gibson şi Randy Katz, o echipă de cercetători de la Universitatea Berkley din California. Tehnologia RAID a fost prezentată la acea vreme ca fiind o metodă ieftină şi eficientă de backup. Aceștia au studiat posibilitatea utilizări a două sau mai multe unităţi ca una singură pentru sistemul gazdă şi au publicat o lucrare intitulată: „Un caz de matrice redundantă de discuri ieftine (RAID)”. Lucrarea lor specifica un număr de niveluri RAID”, fiecare având avantaje şi dezavantaje teoretice.

De-a lungul anilor au apărut diverse implementări ale conceptului RAID. Singurul aspect care a rămas la fel este numerotarea. Majoritatea diferă de nivelul original RAID. Acest lucru poate crea confuzie, deoarece, de exemplu, una dintre implementările RAID 5poate fi complet diferită de cealalta. RAID 3 şi RAID 4 sunt adesea confundate şi chiar folosite interschimbat.

În lucrarea „*Un caz de matrice redundantă de discuri ieftine (RAID)*” autorii defineau formal nivelele RAID de la 1 la 5 în secţiunile 7 - 11:

- „**Primul nivel** RAID: Discuri Oglindite”
- „**Nivelul doi** RAID: Coduri Hamming pentru Corectarea Erorilor”
- „**Nivel trei** RAID: Un Singur Disc De Verificare Per Grup”
- „**Nivelul patru** RAID: Citiri şi Scrieri Independente”
- „**Nivelul cinci** RAID: Date împărţite/paritate pentru toate discurile(nu este un disc unic de redundanţă)”
- **„Nivelul şase** RAID: Redundanţă P+Q”. Aici matricea RAID necesită accesarea a şase discuri datorită necesităţi de a reînnoi ambele informaţii: P şi Q
- **„Nivelul zece** RAID: striped mirrors (oglinzi întreţesute)”**.** Termenul este acum folosit pentru a exprima combinaţia dintre RAID 0 (întreţesut) şi RAID 1 (oglindit).[3]

1.2 Tehnologia RAID

O singură unitate logică formată din mai multe hard discuri fizice ce folosește o componentă hardware sau o aplicație software poartă numele de tehnologie RAID.

Principale tipuri în RAID sunt:

- **mirroring** (*oglindirea*) – se bazează pe copierea aceluiași set de date pe mai multe discuri. Oglindirea poate creşte viteza de citire a datelor deoarece sistemul poate accesa date diferite de pe cele două discuri. În același timp însă, scrierea va fi mai lenta dacă sistemul insistă ca ambele discuri să confirme corectitudinea datelor scrise.
- **striping** (*întreţesute*) – constă în împărţirea datelor pe mai multe discuri; Formatul întreţesut este folosit in mod special pentru mărirea performanţei, deoarece citirea secvenţelor de date se face de pe mai multe discuri in mod simultan.
- **error correction** (*cu corectarea erorilor*) - unde discurile redundante de verificare stochează datele pentru a fi detectate şi corectate eventualele erori. Verificarea erorilor în mod obişnuit va încetini sistemul deoarece datele vor fi citite din mai multe locaţii şi apoi comparate. [4]

Diferitele configurații afectează în mod diferit stabilitatea și performanța (viteza de acces). Folosirea mai multor discuri în paralel crește probabilitatea ca unul dintre ele să se defecteze; dar utilizând funcții automate de detectare și corectare a erorilor, sistemul poate deveni mai stabil și poate repara în mod automat(„din mers") anumite erori. Gamele de discuri moderne oferă posibilitatea de a alege și a schimba configurația RAID după dorință.

Sistemele RAID au fost proiectate să ruleze in continuare chiar și în caz de defectare completă a unui disc– discurile pot fi schimbate „la cald" iar datele pot fi recuperate în mod automat, în timp ce sistemul rulează în continuare (eventual un pic mai lent, până la terminarea recuperării datelor). Prin comparație, sistemele de discuri normale trebuie oprite până când datele sunt recuperate.

RAID este adesea folosit în sistemele ce necesită o rată de acces cât mai ridicată și unde este important ca sistemul să ruleze cât mai multă vreme cu putință. În general, RAID este folosit la servere, dar poate fi folosit și în cazul stațiilor de lucru (workstation). [3]

1.3 Implementarea tehnologiei RAID

RAID combină hard discuri fizice într-o singură unitate logică folosind o component hardware sau o aplicație software. Soluțiile hardware prezintă sistemului RAID atașat ca un singur hard disc, fără ca sistemul de operare să cunoască arhitectura fizică. Soluțiile software sunt implementate în sistemul de operare, dar aplicațiile utilizează arhitectura RAID ca o singură unitate.

Distribuţia de date între mai multe unităţi poate fi realizată atât hardware cât şi software. Pe lângă aceste două modalități exista şi un mod hibrid, format atat dintr-o pare hardware, cat si dintr-o parte software.

1.3.1 Implementare software

Majoritatea sistemelor de operare furnizează implementări software. Acest layer se situează deasupra driverelor discurilor şi oferă un strat de abstractizare între discurile logice şi discurile fizice. În mod normal RAID-ul software se limitează la RAID 0, 1 şi 5. În sistemele de operare multi-thread (Linux, Mac OS X, Novell,Windows NT, 2000..) sistemul de operare poate executa mai multe procese I/O (permite scrieri şi citiri multiple). Această caracteristică permite, în plus, realizarea RAID 0/1 în implementare software cu toate că nimeni nu foloseşte această practică. Acest lucru se realizează deoarece calcularea parității (datele redundante) necesită o putere de procesare ridicată. Implementările software necesită timpi reduși de prelucrare și consumă foarte puține resurse. În anul 2007 calculatoarele personale, obişnuite, ofereau puterea de calcul necesară crescând folosirea procesorului cu doar un procent în plus. Ele erau eficiente ca şi cost până într-un punct neoferind însă acelaşi grad de performanţă ca şi RAID-ul bazat pe hardware. RAID-ul software foloseste resursele unui server legat la spaţiul de stocare, CPU-ul acestuia fiind doar partial ocupat cu procesarea datelor. În cazul unei defecţiuni a serverului, datele aflate in spaţiul de stocare nu pot fi accesate atâta timp cât defecţiunea persistă. În hardware matricea de HDD-uri poate fi ataşată uşor altei gazde astfel timpul de recuperare se reduce simţitor .Totuşi acest sistem este la fel de eficient ca şi cel software în cazul în care nu există un sistem de backup al hostului pe care să se facă trecerea rapid.

Implementarea software permite si crearea de matrici RAID din partiţii ale unui HDD.

1.3.2 Implementare hardware

RAID-ul hardware solicită minim un controller RAID. Într-un calculator personal acest controller poate consta intr-un card PCI sau poate fi inclus în chipset-ul plăcii de bază. În aplicaţiile industriale controllerul lucrează ca şi unitate separată. HDD-urile pot fi SATA, IDE/ATA, SSA, SCSI, canal optic sau combinaţii ale acestora. Sistemul ce utilizeaza acest sistem de stocare poate fi conectat in mod direct la controller sau printr-o reţea SAN (StorageArea Netork). SAN-ul este o reţea ce utilizeaza un protocol de comunicare asemănător cu SCSI. Acest protocol se poate suprapune peste o reţea Ethernet. Controlerul se ocupa de managementul discurilor şi calculează datele redundante caracteristice nivelului RAID ales. Majoritatea acestor sisteme au implementată o memorie cache nevolatilă, ce îmbunătățește performanţele. Implementarea hardware prezintă performanţe garantate, fără utilizarea procesorului gazdei, controllerul prezentând sistemului de operare un simplu disc logic. De asemenea, sistemele industriale prezintă suport pentru "hot swapping" (schimbarea HDD-urilor defecte fără a fi necesară oprirea sistemului).

1.3.3 Implementare hibridă

Sistemele hibride RAID si-au facut apariția o dată cu introducerea controllerelor ieftine RAID implementate în controlere HDD ca și extensii de BIOS reprezentate de drivere de sistem. Din păcate aceste controllere efectuează toate calculele necesare în regim software. Ele adună majoritatea dezavantajelor implementării software şi hardware. Ca şi implementările hardware ele sunt proprietare unui fabricant de controllere RAID şi nu se pot face combinări între mai multe controllere. Avantajele constau în abilitatea de a boot-a de pe discul logic şi integrarea mai strânsă cu driverul device-ului ce oferă o manipulare mai bună a erorilor.

CAPITOLUL 2

Niveluri RAID Standard

Iniţial au fost concepute 5 niveluri RAID, dar pe măsură ce tehnica a evoluat au apărut mai multe variaţii ale acestora, cum ar fi nivelurile hibrid şi mai multe niveluri non-standard. Acestea niveluri RAID (inclusiv formatele de date asociate lor) sunt standardizate de către SNIA în standardul DDF – *Common RAID Disk Drive Format.*

Diferitele niveluri RAID folosesc unul sau mai multe dintre tipurile enumerate la punctul 1.2 în funcţie de cerinţele sistemulu. Principalele obiective ale arhitecturilor RAID sunt: mărirea siguranţei datelor şi creşterea vitezei de acces. Stabilitatea şi performanţa sunt afectate în mod diferit de către diferitele configuraţii ce pot fi folosite.

2.1 RAID 0 – Disk Stripping

RAID 0 nu este o arhitectură RAID în adevăratul sens al cuvântului deoarece nu asigură nicio redundanţă a datelor. RAID 0 este folosit strict pentru maximizarea performanţelor în lucrul cu harddisk-urile. După cum spune şi denumirea, datele sunt întreţesute (stripping) în mod secvenţial pe mai multe discuri, ce sunt tratate ca un singur disc (sau volum) virtual. De obicei 4 discuri formează un volum. Implementările RAID 0 divizează volumele în felii şi scriu datele în felii consecutive, localizate fizic pe fiecare disc din cadrul ariei de discuri.

Operaţiile de scriere şi citire au loc în paralel fiind executate pe toate discurile aflate în volum. Mărimea feliilor este definită de către utilizator şi poate fi de 512 sectoare ,spre exemplu, dimensiunea uzuală a sectoarelor putând fi de 512 octeţi. Utilizând această tehnică de întreţesere se obţine o rată de transfer a datelor foarte ridicată, îmbunătăţindu-se în felul acesta şi performanţa efectivă a sistemului.

În eventualitatea defectării chiar şi a unui singur disc al ariei, RAID 0 nu oferă redundanţă datelor. În lipsa posibilităţii de regenerare a discului, datele ce se găsesc pe acesta se vor pierde.

Din acest motiv, RAID 0 nu este folosit în aplicaţiile de înaltă disponibilitate. [5][6]

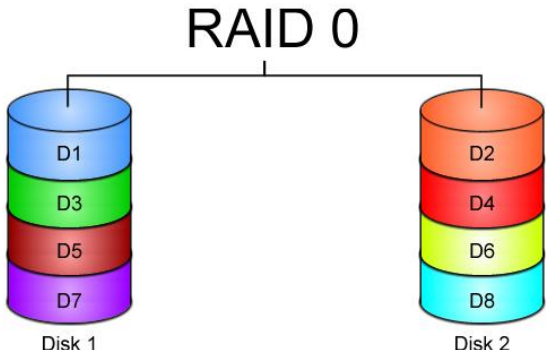

Fig. 2.1

Repartiţia blocurilor de date în RAID 0

2.2 RAID 1 – Disk Mirroring

RAID 1 asigură redundanţa datelor prin oglindirea acestora (mirroring). Metoda aceasta creează două sau mai multe copii ale aceluiaşi volum de date şi le plasează pe discuri separate. Fiecare copie este o imagine oglindită a celeilalte.

RAID 1 poate scădea performanţa generală a sistemului deoarce trebuie să efectueze tot atâtea scrieri câte copii. Performanţa citirilor este însă mult îmbunătăţită deoarece cererile sunt trimise simultan controlerelor de discuri ale tuturor copiilor. Discul care răspunde primul cererii de citire va returna datele sistemului.

Această metodă este mai puţin sigură comparativ cu scrierea secvenţială, dar realizează o creştere semnificativă a performanţelor la scriere. RAID 1 este cea mai scumpă implementare deoarece asigură redundanţa datelor în proporţie de 100% şi totodată asigură cea mai mare disponibilitate a datelor prin utilizarea celui mai mic număr de discuri din configuraţie.

Anumite sisteme de operare de tip UNIX ale unor sisteme de calcul tolerante la defectări oferă prin intermediul unei componente software numită "*Logical Volume Manager*" un alt tip de RAID 1. Această componentă creşte performanţa sistemului oferind posibilitatea scrierii în paralel pe toate copiile, fără a se mai aştepta ca scrierea pe o copie să se efectuze cu success şi apoi să se altereze conţinutul altei copii. Comparativ cu scrierea secvenţială, această metodă este mai puţin sigură, dar oferă o creştere semnificativă a vitezei de scriere. Întrucât RAID 1 asigură redundanţa datelor în proporţie de 100%, ea este cea mai scumpă implementare dintre toate tipurile de arii de discuri, dar asigură şi cea mai mare disponibilitate a datelor prin utilizarea celui mai mic număr posibil de discuri în configuraţie. [5][6]

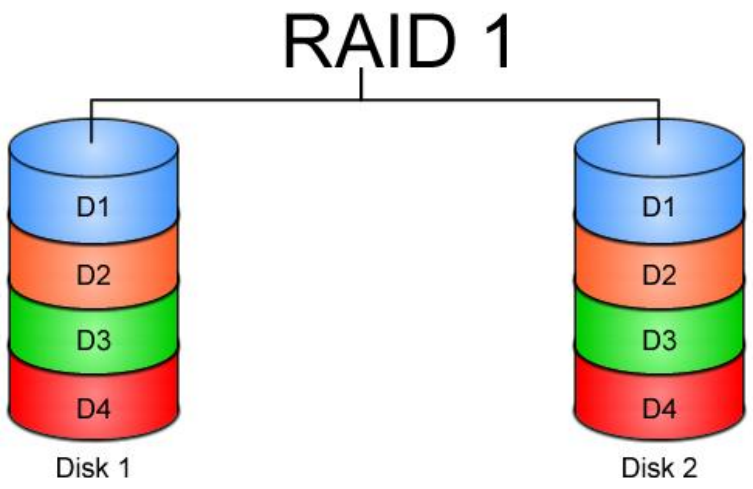

Fig. 2.2
Repartiţia blocurilor de date în RAID 1

2.3 RAID 2 – Parallel Array with Error Corection Code

Acest model presupune că discurile în paralel distribuie datele pe mai multe discuri efectuând o întreţesere la nivel de bit, iar în cadrul ariei de discuri sunt utilizate şi discuri de verificare. Citirea şi scrierea datelor se face utilizând tehnici de codificare bazate pe coduri corectoare de erori de tip Hamming pentru a asigura detecţia şi corecţia erorilor.

Din păcate RAID 2 vine cu mai multe dezavantaje decât avantaje. Printre acestea trebuie amintit faptul că utilizarea tehnicilor de codare bazate pe coduri corectoare de erori de tip Hamming necesită utilizarea unor grupuri mari de discuri pentru a asigura consistenţa (ex: 4 discuri de verificare pentru 10 discuri de date, 5 discuri de verificare pentru 25 de discuri de date). Din cauză că această tehnică de codificare este foarte complexă şi scump de realizat, RAID 2 nu prezintă un real interes pentru mediul comercial.

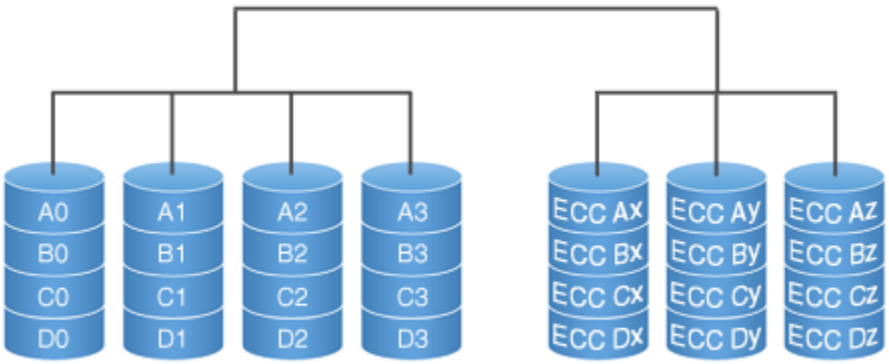

Fig. 2.3
Repartiţia blocurilor de date în RAID 2

2.4 RAID 3 – Parallel Array With Parity

RAID 3 foloseşte doar un singur disk dedicat pentru memorarea informaţiilor de paritate, celelalte discuri din cadrul ariei de discuri fiind folosite pentru memorarea informaţiei utile ce este împrăştiată pe toate discurile ariei. (exact ca şi în cazul RAID 2).

Acesta efectuează un SAU EXCLUSIV asupra datelor şi nu un cod corector de erori.

Cantitatea minimă de date care este scrisă sau citită într-o operaţie de intrare sau de ieşire este egală cu numărul de discuri din arie înmulţit cu numărul de octeţi ai sectorului discului. Aceasta este cunoscută ca unitate de transfer. La un moment de timp dat nu poate fi activă decât o singură cerere de intrare sau de ieşire deoarece capetele de acces se mişcă în paralel în aria de discuri. Acest lucru ne asigură o rată bună de transfer a datelor atunci când datele accesate sunt dispuse secvenţial pe disc, dar face ca RAID 3 să nu fie suficient de practic pentru aplicaţii cu date dispuse aleator în aria de discuri. O rată de transfer foarte bună se obţine atunci când unitatea de transfer are aceaşi mărime cu dimensiunea blocurilor de date care se scriu sau se citesc. Cu cât blocurile transferate sunt de dimensiuni mai mici, cu atât rata de transfer scade.

RAID 3 oferă si posibilitatea înlocuirii la cald a unui disc şi recuperarea datelor. [7][8][9]

2.5 RAID 4 – Striped Array with Parity

În cazul RAID 4 datele sunt distribuite la fel ca şi în cazul RAID 3 pe mai multe discuri de date şi pe un disc de paritate. Diferenţa constă în faptul că un bloc de date se află pe un singur disc, astfel încât citirea se face accesând un singur disc şi nu pe ambele. Astfel aria de discuri poate prelucra mai multe cereri deodata întrucât discurile nu se mai utilizează în paralel.

Discul de paritate devine un element de scădere a performanţei deoarece este implicat în toate operaţiile de scriere. Acest lucru se întâmplă deoarece paritatea, pentru fiecare operaţie de scriere, se calculează având în vedere şi informaţia de pe celelalte discuri, în sectoarele de date asociate. Această procedură face ca RAID 4 să fie considerat nepractic. [9]

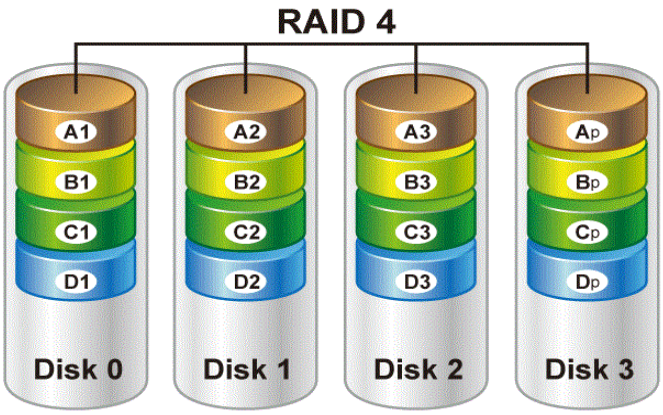

Fig. 2.4
Repartiția blocurilor de date și de paritate în RAID 4

2.6 RAID 5 – Striped Array with Distributed Parity

În cazul RAID 5 datele sunt distribuite pe mai multe discuri de date şi pe un disc de paritate. RAID 5 nu are însă un disc dedicat special numai pentru informaţia de paritate, ci distribuie informaţiile de tip date şi informaţiile de tip paritate pe toate discurile ariei.

RAID 5 permite mai multe accese concurente la dispozitivele din cadrul ariei de discuri, fiind satisfăcute astfel multiple cereri concurente de intrare sau ieşire. Astfel, performanţa transferurilor pentru aceste arii de discuri creşte semnificativ, acestea devenind cele mai potrivite şi utile în cazul operării cu blocuri de dimensiune mică, sau cu datele dispuse în mod aleator pe discuri.

Ciclul de scriere se realizează în 3 paşi:

- Pasul 1 – Este citită paritatea existentă şi data ce urmează a fi schimbată
- Pasul 2 – Vechea dată din valoarea parităţii existente este scazută şi apoi se memorează paritatea corectă într-un buffer
- Pasul 3 – Este calculată noua paritate pe baza valorii din buffer şi a valorii noii date de memorat şi se scriu pe discurile corespunzatoare noua dată şi paritatea actualizată.

Implementările software ale RAID 5 sunt suportate de către multe sisteme moderne de operare (Linux, Windows Server etc.). Procesorul efectuează toate calculele pentru paritate, iar în cazul unei defecţiuni, operaţiunile de intrare sau de ieşire sunt de până la de 3 ori mai consumatoare de resurse. Având un cost relativ scăzut per GB utili şi o performanţă medie foarte bună, RAID 5 este utilizat cu succes pentru toate tipurile de transfer (servere pentru baze de date raţionale, servere de fişiere) [1][3][9][10]

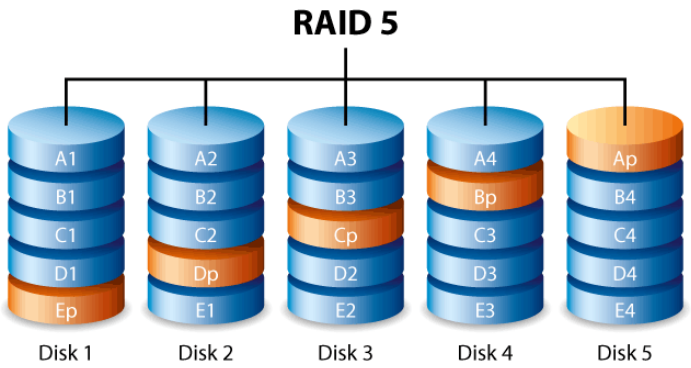

Fig. 2.5a
Repartiția blocurilor de date și de paritate în RAID 5

2.7 RAID 6 – Striped Array with Dual Distributed Parity

Ceea ce diferenţiază nivelul RAID 6 de nivelul RAID 5 este faptul că se calculează şi se stochează paritate dublă pentru date. În consecinţă sunt necesare minim 4 discuri, performanţa scăzând datorită calculului dublu de paritate. Redundanţa creşte însă, aria suportând două discuri defecte. Reconstrucţia datelor poate fi un mare consumator de resurse, performanţa fiind puternic afectată, dar redundanţa dublă compensează prin posibilitatea de amânare a procesului de rebuilding pentru un plus de viteză.

Pentru a crea nivele etajate de RAID, oricare nivel poate fi combinat cu alt nivel, dar cel mai des sunt folosite nivelele 0 şi 1. [11][12]

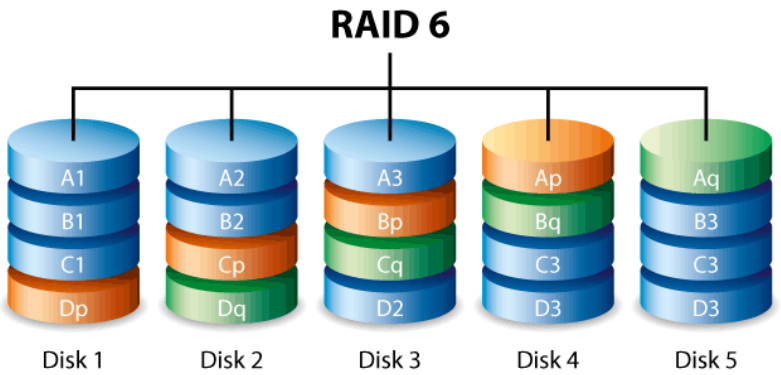

Fig. 2.5b
Repartiția blocurilor de date și de paritate în RAID 6

Niveluri RAID Hibrid

Nivelurile RAID hibride combină două sau mai multe niveluri standard de RAID pentru a obţine performanţă şi redundanţă suplimentară. Atunci când combinăm nivelurile RAID, un tip de RAID care oferă redundanţă este, de obicei, unit cu un tip de RAID 0 ce mărește performanţa. Cu aceste configuraţii, este de preferat să existe RAID 0 pe partea de sus şi matrici redundante în partea de jos, deoarece mai puţine discuri vor trebui să fie regenerate atunci când un disc se defectează. Prin urmare, RAID 1+0 este preferabil lui RAID 0+1, dar avantajele administrative ale "oglinzii împărţite" de RAID 1 ar fi pierdute. Trebuie remarcat, cu toate acestea, faptul că aspectul blocurilor pe disc pentru configuraţiile RAID 1+0 şi RAID 0+1 sunt oarecum identice, deci aceste limitări apar doar în software. [13]

2.8 RAID 01 (0+1)

Acest nivel dispune minim 4 discuri grupate în fâșii (oglindite), înlătură neajunsurile şi îmbunătățește performanţele, însă crește complexitatea. RAID 0+1 creează un al doilea set de fâşii oglindite cu un set primar de fâşii de discuri. Matricea continuă să funcţioneze cu unul sau mai multe unităţi de stocare afectate în aceeaşi oglindă stabilită, dar dacă driverul cedează de pe ambele părţi ale oglinzii, datele de pe sistemul RAID sunt pierdute. [2][3]

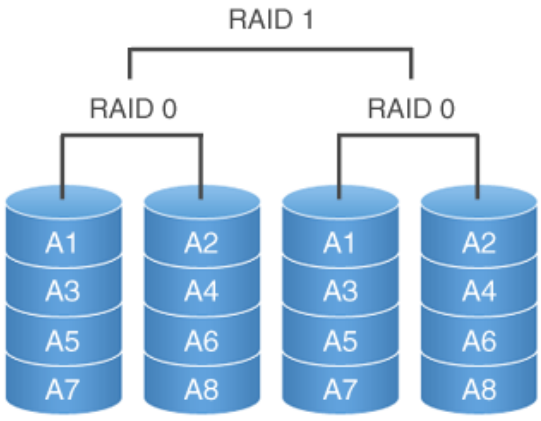

Fig. 2.6
Repartiţia blocurilor de date: RAID 0+1

2.9 RAID 10 (1+0)

Acest nivel dispune minim 4 discuri grupate în fâșii (oglindite), înlătură neajunsurile și îmbunătățește performanțele, însă crește complexitatea. Diferența față de RAID 0+1 este că RAID 1+0 creează discuri cu date întreţesute de la o serie de unităţi de stocare în oglindă. Într-o situaţie de eşuare, RAID 1+0 funcţionează mai bine, deoarece toate celelalte discuri continuă să fie utilizate. Matricea poate susţine mai multe discuri pierdute, atât timp cât oglinda nu pierde toate unităţile sale. [2][3]

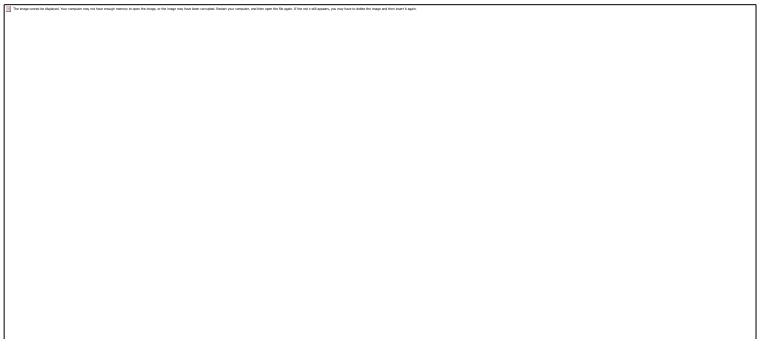

Fig. 2.7a
Repartiţia blocurilor de date: RAID 1+0

2.10 RAID 100 (10+0)

Un raid 100, numit uneori și RAID 10+0, este o felie de tipuri RAID 10. Din punct de vedere logic el este de fapt o matrice RAID 10 implementată cu ajutorul software-ului RAID 0 peste hardware-ul de RAID 10.

Principalul avantaj al RAID 100 față de un singur nivel RAID este răspândirea încărcării pe mai multe controllere, obținându-se astfel performanță mai bună la citiri aleatoare și atenuarea riscurilor pe matrice. Tocmai din acest motiv, RAID 100 reprezintă cea mai bună alegere când vine vorba de baze de date foarte mari, unde controllerele hardware de RAID limitează numărul de discuri fizice permise în fiecare matrice standard. [2][3]

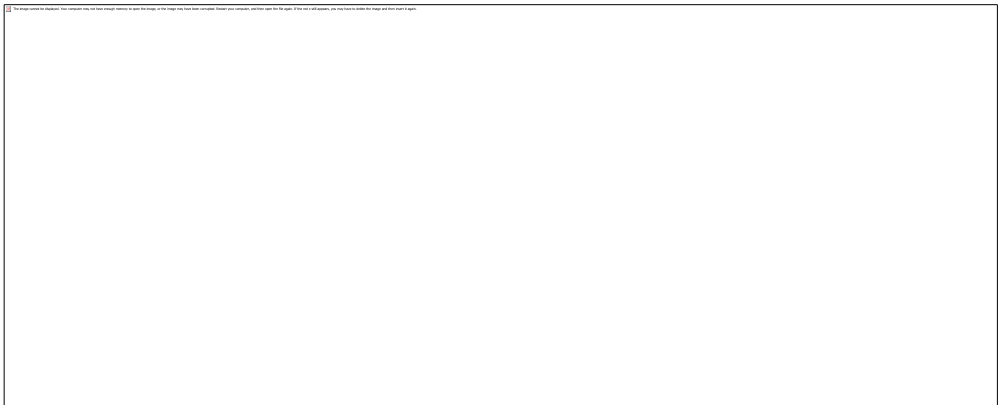

Fig. 2.7b
Repartiția blocurilor de date: RAID 10+0

2.11 RAID 51 (5+1)

Acest nivel hibrid este o matrice formată din două matrice RAID 5, care sunt oglindite una față de cealaltă. În general această configurație este folosită astfel încât fiecare set de RAID 5 se află pe un controller separat. Astfel, scrierile și citirile sunt echilibrate în ambele matrice RAID 5. Unele controllere suportă RAID 51 pe mai multe canale pentru a ține sincronizate părțile diferite. În această configurație, cele două seturi de RAID 5 nu știu că reprezintă oglindirea celuilalt, iar RAID 1 nu știe că discurile sale de bază sunt RAID 5.

Această configurație poate susține eșecul oricărui disc din orice matrice, plus încă un disc suplimentar din cealaltă matrice înainte să existe pierderi de date. Spațiul maxim ce se poate găsi pe un RAID 51 este egal cu dimensiunea unui set individual RAID 5. [13][2][3]

2.12 RAID 50 (5+0)

RAID50 combină feliile de blocuri de la nivelul RAID0 cu paritatea distribuită de RAID5 Aceasta este de fapt o matrice RAID 0 întrețesută de-a lungul elementelor RAID 5. Pentru a realiza un nivel hibrid RAID 50 avem nevoie de cel putin 6 discuri.

Avantajul acestei configurații este că ca pot eșua câte un drive din fiecare set, fără să se piardă date. Cu toate acestea, unitățile rămase în acel set devin un punct slab pentru întreaga matrice până când discul defect este înlocuit. Este suficient să se mai defecteze un singur disc (pe lângă cel inițial) și întreg sistemul cedează, datele stocate în întreaga matrice fiind pierdute. Timpul petrecut în procesul de rebuilding reprezintă o perioadă de vulnerabilitate a setului RAID. [13][2][3]

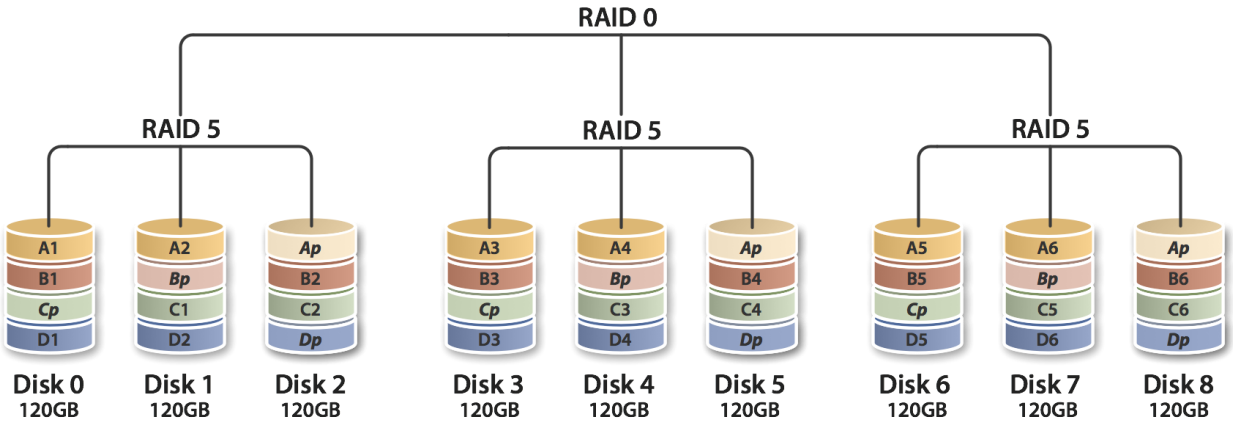

Fig. 2.8
Repartiția blocurilor de date și de paritate: RAID 5+0

Avantaje și dezavantaje

RAID	Avantaje	Dezavantaje
0	Ușor de proiectat și implementat Nu folosește calculi de paritate	Nu tolerează greșeli. Defectarea unui singur disc duce la pierderea tuturor datelor dintr-un șir.
1	Viteza de citire este de 2 ori mai mare Redundanță 100% Poate susține simultan mai multe defecțiuni În caz de defectare datele nu trebuie reconstruite ci pur și simplu copiate	Capacitatea efectivă de memorie este de fapt jumătate din capacitatea maximă totală Posibil să nu suporte schimbarea discului în timpul funcționării
2	Rate de transfer foarte ridicate	Rata de transfer este cel mult egală cu cea a unui singur disc
3	Viteză mare de citire și scriere pentru pachetele mari de date Disk-failure nu încetinește ratele de transfer	Tehnologia este complexă și mare consumatoare de resurse, deci realizarea software a acesteia nu se rentează
4	Transfer de date foarte mare la citire Eficiență ridicată datorită discurilor de paritate aflate în proporție mai mică decât a celor de date	Cea mai slabă rată de transfer la scriere În cazul defectării unui disc, recuperarea datelor este extreme de dificilă
5	Cea mai ridicată rată de transfer la citire Rată de transfer bună la scriere	Datele sunt dificil de recuperat în comparație cu nivelul RAID 1
6	Toleranță foarte mare la erori Poate susține mai multe defectări simultane	Schema de paritate este bidimensională deci trebuiesc implementate N+2 discuri
01	Înlătură neajunsurile și îmbunătățește performanțele	Crește complexitatea Dacă driverul cedează pe ambele părți ale oglinzii, datele sunt pierdute
10	Toleranță la erori. Poate susține mai multe defectări simultane, în anumite cazuri	Tehnologie foarte scumpă Performanța scade întrucât discurile se mișcă în paralel la adresa cerută
50	Un drive de la fiecare din seturille RAID5 poate eșua fără pierderi de date	Sistemul rămâne vulnerabil până la schimbarea discului căzut
51	Poate susține eșecul fiecărui disc din matrice plus încă un disc suplimentar Performanță bună dar costisitoare în același timp	Costul foarte mare al implementării din cauza discurilor identice ce trebuiesc folosite și a controllerelor de ultimă generație
100	Răspândirea încărcării de-a lungul mai multor controller Performanță mai bună la citiri aleatoare	Complexitate foarte ridicată Prețul foarte ridicat al implementării

CAPITOLUL 3

Rebuilding

3.1 Noțiuni Generale

Noţiunea de rebuilding se referă la reconstrucţia sistematică a datelor de pe discul defect, pe un disc de rezervă. Aceasta procedură începe imediat ce discul de rezervă devine disponibil.

Matricile RAID folosesc discuri de rezervă pentru a minimiza pierderea datelor. Astfel după ce un disc se defectează, datele de pe acesta pot fi reconstruite imediat pe discul de rezervă (rezervă dedicată – "dedicated sparing"). Rezerva distribuită – "distributed sparing" şi rezerva de paritate – "parity sparing" reprezintă două alternative de rezervă. În cazul rezervei de paritate sistemul porneşte fără rezervă, dar dacă un disc pică, atunci grupurile de paritate aflate în matrici diferite sunt combinate pentru a forma matrici şi grupuri de paritate mai mari.

O matrice RAID poate funcţiona în unul din următoarele 3 moduri:

- **Modul Normal** – Discurile principale sunt operaţionale, iar discul de rezervă poate fi atât funcţional cât şi defect sau în reparaţie.

- **Modul Deteriorat** – Presupune deteriorarea unuia din cele N+1 discuri primare. În continuare, pentru a recrea blocul de date pierdut, cererea pentru citirea datelor de pe discul defect presupune accesul la blocurile corespunzatoare de pe discurile funcţionale. O cerere de scriere pe discul eşuat rezultă în citirea celor N blocuri de date corespunzătoare pentru a calcula blocul de paritate (folosind noua versiune a blocului de date), urmată de o simplă scriere a blocului de paritate. Astfel, rata de cereri pentru discurile funcţionale aproape se dublează.

- **Modul Rebuild** – Operatia de rebuilding porneşte după ce unul din discuri iese din funcţiune, nemai fiind capabil să reintre în modul normal de lucru al sistemului. În modul de rebuild sistemul citeşte trasee consecutive de date de la toate discurile funcţionale în paralel, calculează conţinuturile traseelor discului defect (prin aplicarea operaţiei XOR asupra celor N trasee coresupunzatoare din discurile funcţionale), şi scrie traseele reconstruite pe discul de rezervă. (În scrierea pe discul de rezervă nu luăm în considerare opţiunea de a verifica, conform căreia blocurile de date abia scrise sunt recitite pentru a verifica scrierea. Această opţiune aproape dublează încărcarea pe discul de rezervă, astfel încât timpul de rebuild este de aşteptat să fie dominat de timpul de scriere pe acest disc când această opţiune este activă). Sistemul este predispus la pierderea datelor în cazul în care unul din discurile funcţionale cedează înainte ca reconstrucţia să fie completă.

Cea mai mică unitate de date ce poate fi reconstruită se numeşte Unitate de Rebuild (UR) şi este de obicei o unitate întreţesută sau o parte a acesteia.

$T_{rebuild}$ reprezintă timpul necesar reconstruirii discului defect, iar $T_{răspuns}$ reprezintă timpul de răspuns la cererile utilizatorului. Aceste două mărimi sunt principalele repere pentru măsurarea performanţei sistemului.

Rebuildingul se împarte în 2 categorii: orientat pe fâşii şi orientat pe disc. În cazul rebuildingului orientat pe disc se dedică un proces pentru fiecare disc care citeşte UR în mod asincron de pe discurile ce au supravieţuit. În rebuildingul orientat pe fâşii, fiecare UR este reconstruită printr-un proces dedicat care citeşte UR de pe discurile supravieţuitoare, execută între ele operaţia logică XOR şi scrie UR rezultată pe discul de rezervă.

Rebuildingul orientat pe disc necesită un buffer mai mare faţă de rebuildingul orientat pe fâşii atunci când se consideră un număr mic de procese, însă rebuildingul orientat pe disc este mai performant.

Din punct de vedere al spaţiilor tampon se poate afirma că există 2 categorii: spaţii tampon temporare şi spaţii tampon dedicate scrierii.

Spaţiile tampon temporare sunt dedicate citirilor de pe fiecare disc, în timp ce spaţiile tampon dedicate scrierii sunt dedicate scrierii pe discul de rezervă.

Spaţiile tampon sunt interschimbabile şi au ca unitate de măsură tot o unitate de reconstrucţie (UR). Imediat ce o UR este citita in spatial tampon, i se aplica o operatie logica XOE cu unitatile UR corespunzatoare iar rezultatul este stocat în zona tampon de pe discul de rezervă. Spaţiile tampon nu se epuizează niciodată deoarece operaţia XOR este foarte rapidă. Datorită acestui fapt, citirea unităţilor de UR de pe discuri, în timpul unui rebuild orientat pe disc, nu este oprită din cauza limitărilor de memorie. Totuşi, spaţiul tampon pentru discul de rezervă, are un efect asupra performanţei de reconstrucţie. Toate trimiterile viitoare la spaţiul tampon implică memoria tampon a discului de rezervă.

Cererile de rebuild pot fi procesate în două feluri: cu aceaşi prioritate ca şi cererile utilizatorului şi la o prioritate mai mică decât cererile utilizatorului. **Modelul Clientului Permanent** (PCM) presupune o prioritate egală a cererilor de rebuild cu cererile de utilizator. În acest model o singură cerere este procesată la un moment dat într-un sistem de tip coadă de aşteptare. Imediat ce o cerere a fost servită, îi vine rândul următoarei, iar la sfârşitul cozii de aşteptare vine o alta nouă.

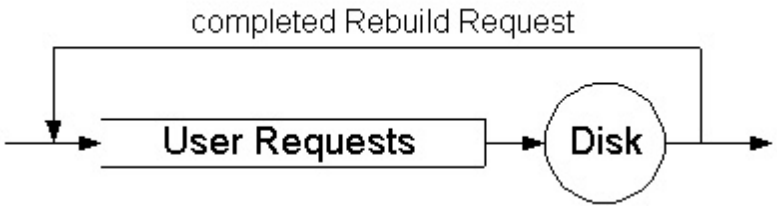

Fig. 3.2
Modelul PCM de procesare a cererilor

Modelul Serverului de Eliberare (VSM) reprezintă o alternativă mai bună pentru PCM, oferind timpi de răspuns şi de reconstrucţie mai mici. Acesta efectuează reconstrucţia atunci când nu se află nicio cerere de utilizator în aşteptare (adică imediat ce discul devine inactiv). Imediat ce soseşte o cerere de utilizator reconstrucţia este oprită.

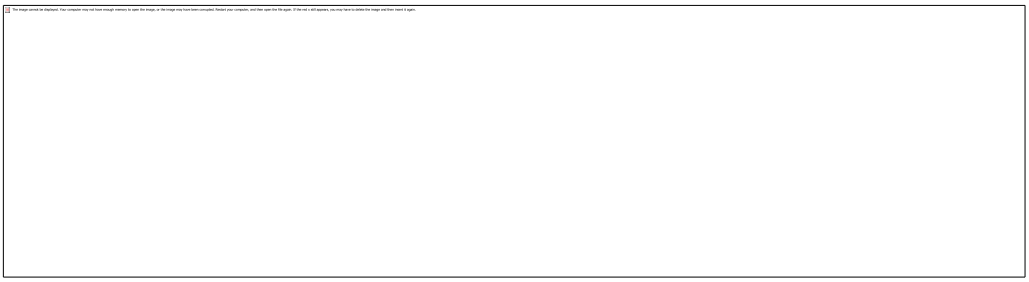

Fig. 3.3
Modelul VSM de procesare a cererilor

3.2 Functionarea matricilor RAID 5

Un sistem RAID 5 este format din N+1 discuri primare care conţin blocuri de date, blocuri de paritate şi un disc de rezervă dedicat. Un şir de biţi asociaţi pe blocurile de date folosind operaţia logică XOR formează un bloc de paritate. Blocurile de paritate sunt distribuite uniform în discurile primare pentru a preveni o posibilă ştrangulare.

Pentru a scrie un bloc de date, matricile RAID 5 necesită 4 accesări ale discului pentru că paritatea trebuie alterată de fiecare dată când blocurile de date sunt modificate.
Cele 4 accesări ale discului sunt:

1. Citire date vechi
2. Citire paritate veche
3. Scrierea noilor date
4. Scrierea noii paritati

Înainte de a începe recuperarea datelor de pe o matrice RAID 5 trebuie să determinăm configuraţia şi parametrii. Configuraţia unei matrice RAID 5 constă în cunoaşterea următorilor parametrii:

- Cate discuri sunt in configurația RAID 5
- Care este secvența discurilor
- Ce valoare s-a folosit pentru mărimea blocurilor
- Care este pattern-ul de paritate folosit

Toţi aceşti parametrii pot fi determinaţi manual sau automat. Pentru a determina poziţia parităţii trebuie să ştim faptul că rotaţia acesteia decurge după cum este ilustrat în figura urmîtoare:

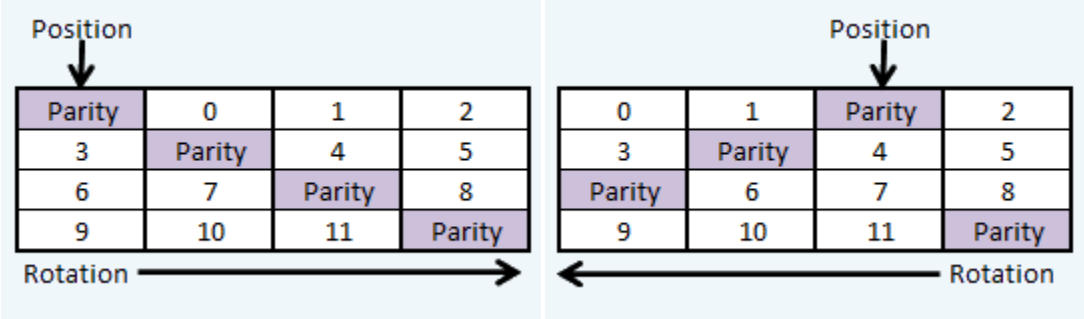

Fig. 3.4
Rotația parității

Pentru a determina poziţia trebuie să privim blocurile de memorie din matrice. Blocul care arată cel mai puţin ca un bloc de date, este blocul de paritate.

Determinarea ordinii discurilor se realizează prin urmărirea fişierelor text foarte lungi. Se preferă acele fişiere care au şi timestamps. Pentru a găsi astfel de fişiere pe discuri se folosesc tool-uri special, cum ar fi WinHex. Când s-a găsit un fragment dintr-un astfel de fişier pe unul din discuri, este nevoie să găsim discul ce conţine următorul fragment şi aşa mai departe. În acest fel se poate determina ordinea discurilor, deşi este imposibil de afirmat care disc a fost primul.

Pentru a determina care disc a fost primul, trebuie să folosim din nou un tool de vizualizare a discurilor şi să căutam următoarele 2 lucruri, în funcţie de tipul de RAID pe care îl avem:

- MBR-ul (Master Boot Record) – in cazul unui RAID realizat hardware
- Sectorul de boot – in cazul unui RAID realizat prin mijloace software

În cazul unui RAID realizat hardware, discul care conţine MBR-ul este primul disc din acel RAID. *Master Boot Record* este un tip special de sector de boot aflat la începutul partiţiilor unui element masiv de stocare (*mass storage device*). MBR-ul reţine informaţii cu privire la organizarea partiţiilor logice pe acel mediu de stocare. Partiţiile logice conţin fişierele de sistem.

În cazul unui RAID realizat software, discul ce conţine Sectorul de Boot la început este primul disc. [14]

3.3 Proceduri de rebuilding

Un hard poate întâmpina 2 tipuri de probleme, şi anume:

- probleme logice: problemele apar la nivelul partiţiilor şi pot fi caracterizate prin: coruperea sistemului de fişiere din cauza atacului cu viruşi, ştergerea accidentală a unui folder sau chiar a întregii partiţii, formatarea accidentală şi blocarea sistemului de operare prin intermediul sectoarelor bad. Recuperarea datelor în acest caz nu ridică mari probleme şi presupune utilizarea unor soft-uri concepute special pentu a putea accesa datele corupte. Mai trebuie menţionat faptul că recuperarea datelor cu ajutorul unui soft este valabilă atâta timp cât hard disk-ul este funcţional.

- probleme fizice: se produc atunci când cel puţin una dintre componentele hardului se defectează.

Problemele fizice includ:

- probleme de natură firmware: nu ţin de partea electrică şi pot fi definite ca fiind drept imposibilitatea de a accesa o informaţie aflată pe suprafaţa hardului utilă în citirea datelor existente pe acesta.
  Recuperarea datelor se face cu echipament specializat dar şi cu o vastă experienţă în domeniu.

- probleme mecanice: apar datorită utilizării îndelungate a produsului, fluctuaţiilor de curent electric sau a diferitelor probleme legate de fabricaţia acestuia. Când apare un defect mecanic următoarele componente pot fi afectate: stricarea totală sau parţială a capetelor de citire sau scriere, deteriorarea motorului, zgârierea suprafeţelor.
- probleme electrice: apar mai ales datorită fluctuaţiilor de curent. Numai cu ajutorul procesului de reconstructie a zonelor afectate se poate ajunge la recuperarea datelor aparent pierdute. [15]

Sistemele RAID pot fi concepute să ruleze mai departe chiar şi în caz de defectare completă a unui disc dur din RAID – discurile pot fi înlocuite „la cald” şi datele recuperate automat, în timp ce sistemul rulează în continuare (eventual ceva mai lent, până la terminarea recuperării datelor).

În procesul de rebuild există 3 proceduri de copiere:

1. **Rebuild folosind procedura de bază** - Actualizările de blocuri de date sau paritate vizate pentru discul defect sunt utilizate pentru a actualiza blocurile discului de rezervă dacă blocul

este deja reconstituit. Pentru a reconstrui un bloc de date, cererile de citire adresate discurilor defecte conduc la citirea blocurilor corespunzatoare din toate discurile încă funcţionale, chiar dacă blocul de date a fost deja recreat în prealabil pe discul de rezervă.

2. **Rebuild cu citire redirecţionată** – În cazul procedurii de bază, blocurile de date sunt recreate. În această a doua procedură, blocurile de date reconstruite pe discul de rezervă sunt citite direct de pe el. Acest lucru ajută la scurtarea timpului de rebuild şi la reducerea utilizării discurilor funcţionale. Chiar dacă citirile redirecţionate interfereaza cu scrierile de rebuild pe discul de rezervă, atâta timp cât discul de rezervă nu constituie o ştrangulare, încheierea procesului de rebuilding depinde în timp de citirea discurilor funcţionale.

3. **Rebuild Piggy-backing pe un workload normal** – "Ideea aici este de a "captura" un bloc de date de pe discul defect, care este reconstruit datorită unei cereri de citire, ce a fost emisă ca parte din workload-ul normal. O implementare relativ simplă este posibilă presupunând doar o uşoară modificare a unui spaţiu tampon convenţional. Atunci când o citire de pe discul defect este satisfăcută de reconstrucţia blocului, conţinuturile blocului vor fi stocate într-un spaţiu tampon. În acest moment copia tampon poate fi marcată ca fiind "modificată", adresa discului asociată cu pagina de tampon poate fi schimbată pentru a referi discul în aşteptare, iar harta de bit pentru reconstrucţie poate fi schimbată pentru a afişa această pagină ca fiind copiată. Înlocuirea normală a spaţiului tampon va copia în cele din urmă conţinutul la blocul corespunzător pe discul în aşteptare. (La sfârşitul reconstrucţiei orice bloc rămas în cache poate fi aruncat la discul în aşteptare). Piggy-backing şi citirea redirecţionată pot ajuta la reducerea încărcării pe discurile funcţionale într-o matrice care are un disc defect. Măsura în care acest lucru poate aduce un beneficiu depinde de mai mulţi factori. Pentru aceeaşi rată de reconstrucţie şi aceeaşi rată minimă de acces pentru workload-ul normal, atât citirea redirecţionată cât şi piggy-backing pot fi utilizate pentru a reduce sarcina pe discurile funcţionale. Aşadar, capacitatea de acces I/O fiind disponibilă pe discurile funcţionale, poate fi folosită pentru a susţine fie un workload mai mare în timpul reconstrucţiei sau un rebuild mai rapid." [16]

"Comparaţia timpului de rebuild pentru diferite astfel de scheme se bazează pe utilizarea valorilor medii în estimarea timpului necesar de dispozitivul de ştrangulare (discurile funcţionale sau discul de rezervă) pentru a finaliza procesul de rebuild. Un sistem RAID cu clustere (grupuri) are mărimea grupului $G \leq N + 1$ şi o rată de declustering $\alpha = (G - 1) / N \leq 1$. Performanţa sistemului în modul deteriorat este mai bună pentru valori mai mici ale lui , din moment ce un număr mai mic de discuri va fi implicat în prelucrarea cererilor fork-join. Una din concluziile studiului care tratează acest aspect este că unele dintre optimizările propuse au ca rezultat o creştere, şi nu o scădere a timpului de rebuild. Acest lucru se datorează faptului că studiul de simulare ia în considerare caracteristicile detaliate de disc, cum ar fi necesitatea de a invoca o urmărire înainte de unele accesări de disc." [17]

CAPITOLUL 4

Simulatorul DiskSim

4.1 Prezentarea simulatorului DiskSim 4.0

Programul DiskSim este un software de simulare eficient, exact şi extrem de configurabil al sistemelor cu discuri. Acest program a fost conceput pentru a fi de folos celor care lucrează în cercetare oferind detalii cu privire la numeroase aspecte din arhitectura subsistemelor de stocare. DiskSim (ajuns la versiunea 4.0) a fost creat de către John S.Bucy, Jiri Schindler, Steven W. Schlosser, Gregoiy R. Ganger, Bruce R. Worthington şi Yale N. Patt la "University of Michigan". Această ultimă versiune (4.0) vine cu îmbunătăţiri aduse variantei precedente (3.0), unele deficienţe fiind rezolvate, precum şi unele librării fiind restructurate pentru o mai mare eficienţă şi pentru a face simulatorul mult mai uşor de utilizat. În particular, modulul de discuri poate simula în mare detaliu configuraţii moderne de discuri fiind recunoscut pentru multitudinea de detalii oferite precum şi pentru fidelitatea acestora.

La nivel structural, simulatorul este compus din mai multe module ce permit configurarea diverselor componente ale unui sistem de stocare. Astfel, au fost făcute configurabile în simulator, discurile, controller-ul, buss-urile, cache-urile precum şi componentele software care dirijează activitatea acestora: driverele, algoritmii de planificare a transferului, organizarea structurilor RAID etc. Simulatorul include de asemenea si un modul micro-electro-mecanic (*MEMS – based module*).

Simulatorul primeşte ca parametru de intrare un fişier cu înregistrări (*trace-uri*) ale activităţii pe magistrala I/O. Aceste *trace-uri* provin de la un sistem real sau pot fi create de un generator intern de sarcini de lucru (workload). Modulul de generare a trace-urilor (generatorul de workload) este foarte flexibil, permiţând modelarea cu acurateţe a diverselor rate de sosire a cererilor. DiskSim 4.0 simulează şi raportează numai aspectele legate de performanţa sistemelor de stocare şi nu modelează comportamentul restului de componente ale unui sistem de calcul sau interacţiunea dintre acestea şi sistemul de stocare pe discuri. Cu toate că simulează aspectele legate de stocarea datelor pe discuri, DiskSim nu salvează sau reface efectiv datele provenite de la o cerere. Partea de cod a simulatorului este scrisă în limbajul C şi necesită pentru compilare compilatorul GCC al sistemului de operare Linux.

Pentru alte aspecte legate de compilarea programului în Linux se va citi fişierul README situat în directorul cu fişiere de cod DiskSim de pe CD-ul anexat prezentei lucrări. Libparam unifică parametrii de intrare ai DiskSim-ului. Acest lucru face mai uşoară simularea utilizând fişierele de input ale modelul de disc dorit fără a mai copia un cod de intrare pentru a putea rula simulatorul. [18][19]

4.2 Utilizarea mediului de simulare DiskSim 4.0

Mediul de simulare DiskSim poate fi încărcat şi compilat pe orice sistem ce rulează Linux, distribuţia acestuia nefiind importantă. Se remarcă o uşurinţă mai mare la instalarea acestuia pe o distribuţie bazată pe Debian.

Pentru a putea rula, DiskSim solicită în linia de comandă 5 parametri. Fişierul curent trebuie stabilit ca fiind "valid".

***DiskSim <parfile> <outfile> <tracetype> <tracefile> <synthgen>* [*par_override*]**

unde:

- **diskSim** – este numele fișierului executabil
- **parfile** – reprezintă numele fișierului cu parametrii
- **outfile** – este numele fișierului de la ieșire. Dacă în locul acestui parametru se aplică ***"stdout "*** rezultatele vor fi afișate pe monitor
- **tracetype** – specifică tipul de trace care se află în fișierul de trace-uri (ascii, validate etc.)
- **tracefile** – este numele fișierului de trace-uri ce va fi folosit la intrare. Pentru ca în loc să folosim un fișier cu trace-uri, putem introduce datele de la tastatură folosind comanda ***"stdin"***
- **synthgen** – setează generatorul de workload-uri în stare activă sau pasivă (valoarea 0 dezactivează generatorul, în timp ce orice altă valoare îl activează).
- **par_override** – permite ca o serie de parametrii implici_i ai simulatorului sau o parte din parametrii transmiși prin fișierul de parametri să fie înlocuiți cu valori transmise prin linia de comandă.

Sintaxa exactă este precizată mai jos:

<component> <parameter> <newvalue>

unde:

- **component** - este numele unei componente ai cărei parametri se doresc a fi modificați;

- **parameter** - este un șir de caractere ce identifică parametrul care va fi înlocuit; Dacă numele conține spații libere el trebuie pus între ghilimele, astfel ca shell-ul să-l considere un singur argument. Pentru a referenția un parametru al unei subcomponente, cum ar fi de exemplu planificatorul unui disc, se folosește scrierea Scheduler parameter:
- **newvalue** - este noua valoare a modulului instanțiat.

4.3 Fișierele de parametri

Pentru a insera fișierul de parametrii, DiskSim folosește *libparam*. Libparam unifică parametrii de intrare ai DiskSim-ului. Intr-un astfel de fișier cu parametrii găsim:

- blocuri – delimitate de accolade {}, ce constă într-un număr de asignări de tipul "name = value"
- instanțieri
- specificații topologice

În exemplu de mai jos este ilustrat un exemplu concret de linie de comandă:

Disksim parm.1B stdout ascii Jan6 0 "disc0" "Segment size fin blks0" 64 "disc*" "Scheduler.'Scheduling policy" 2

Această comandă realizează următoarele acțiuni:

- citește parametrii inițiali din fișierul parms.1B. Specificațiile sunt stocate în fișiere separate și incluse apoi în fișierul principal de parametri, fiind de obicei foarte detaliate.
- exportă rezultatele pe monitor (din cauza comenzii stdout)
- citește din fișierul t.Jan6 fluxul de intrare ASCII
- nu generează activitate sintetică
- se alege planificatorul de disc (în acest caz 2 = Elevator SCAN) pentru discul instanțiat cu numele 'disc0' și pentru driverul acestuia ('driver0')

În fișierul parms.1B există câteva caracteristici ale discului cum ar fi: statistica blocurilor, specificațiile driverelor, magistralelor și ale controllerelor precum și instanțierea acestora sub diferite denumiri. Tot în cadrul acestui fișier, prin comanda "source" este ales fișierul .discspecs și este descrisă topologia sistemului. În fișierul .discspecs este inclus fișierul .model tot prin comanda „source" apoi sunt enumerați parametrii ce se referă la organizarea internă a harddisk-ului (planificator de disc, condiții de scriere / citire etc.). În fișierul .model este descrisă partea cea mai de jos a hard disc-ului

(suprafeţe, cilindri, numărul de blocuri, împărţirea pe zone, piste, sectoare). În ultima parte a acestui fişier se poate alege un fişier .seek din care simulatorul îşi calculează curba de distanţe de căutare. Fişierul .seek ajută la modelarea matematică a harddisk-ului. Specificaţiile sunt de obicei foarte detaliate şi de aceea ele sunt stocate în fişiere separate şi incluse apoi în fişierul principal de parametri.

Fişierul principal de parametri pentru un disc simplu conţine:

- **Max queue length** - numărul maxim de cereri ce pot fi găsite în coadă la un moment dat;
- **Scheduler** - o coadă de I/O;
- **Bus transaction latency** - întârzierea pe care o implică transferul fiecărui mesaj de către disc;
- **Acces time** - sinonim pentru "Constant acces time";
- **Bulk sector transfer time** - timpul necesar discului să transfere pe magistrală un bloc de 512 biţi;
- **Print stats** - specifică dacă statisticile discului sunt sau nu scoase în fişierul de ieşire;
- **Block count** - capacitatea discului exprimată în blocuri;
- **Command overhead** - specifică o întârziere de procesare care are loc imediat ce o cerere a sosit către disc;
- **Constant acces time** - specifică valoarea fixată pentru timpul de acces;
- **Never disconnect** - specifică dacă discul ţine întreg controlul discului pe toată perioada îndeplinirii cererii sau nu (0/1);

4.4 Planificarea accesului la datele de pe disc

Un calculator utilizează cel mai frecvent dispozitivele de I/O, adică discurile. Acest subcapitol prezintă câteva moduri de planificare a accesului la discuri. Un HDD este compus din mai multe platane. Fiecare platan utilizează două capete pentru scrierea şi citirea datelor. Unul dintre capete este folosit pentru partea de sus a platanului, iar celălalt pentru partea de jos. Ambele părţi ale platanului sunt alcătuite din mai multe piste. Fiecare pistă este împărţită la rândul ei în mai multe sectoare.

Un ansamblu de braţe cuprinde capetele ce accesează platanele. Un braț se poate deplasa doar în două direcţii: înspre ax sau direcția opusă axului. Toate capetele de scriere / citire se mişcă înainte şi înapoi împreună, astfel încât fiecare cap este întotdeauna situat la aceeaşi pistă. Nu se poate să avem un cap la pista 0 şi altul la pista 1000. În afară de deplasarea capetelor, axul permite rotirea astfel încât capetele să poată accesa sectorul de pe pista pe care se situează. Din cauza acestui fapt, se întâmplă ca de multe ori locaţia pistei să nu fie specificată ca un număr de pistă, ci mai degrabă ca un număr de cilindru. Un cilindru formează de fapt o colecție a tuturor pistelor. Referirea la sectoare individuale ale discului se face în mod tradiţional prin trimitere la cilindri, capete şi sectoare (CHS). [20]

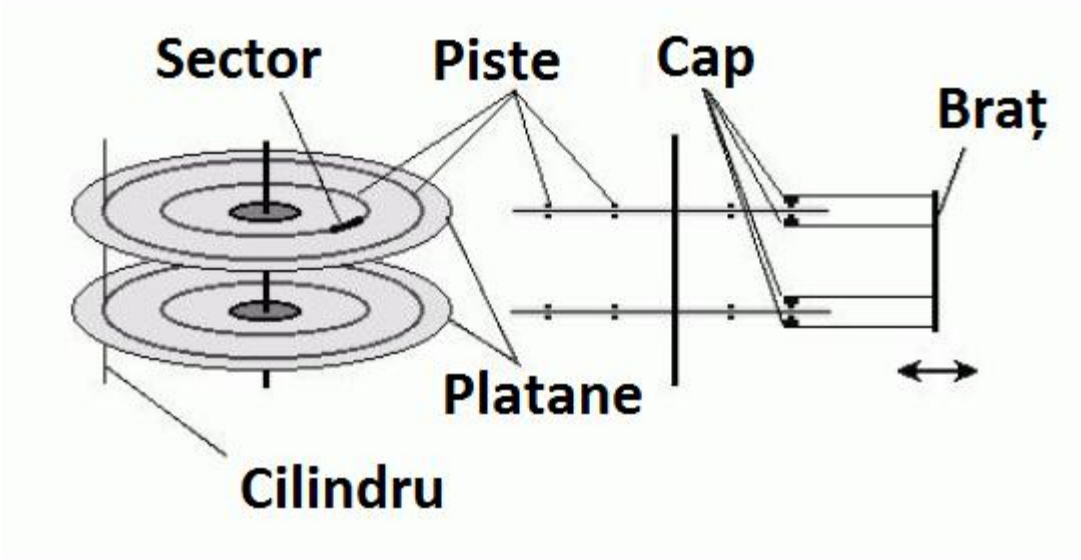

Fig. 4.5
Geometria fizică a hard disk-ului

Atunci când HDD-ul este folosit, discul se învârte la o viteză constantă. În acest fel, o dată ce capul s-a poziţionat deasupra pistei dorite, nu mai trebuie decât să aşteapte până când sectorul dorit se află sub el. Timpul de căutare reprezintă timpul necesar mutării capului, iar timpul necesar pentru ca sectorul dorit să devină disponibil la nivelul capului este cunoscut sub numele de latență rotațională. Operațiile de citire sau scriere pot începe o dată ce capul este în poziția corectă, fapt ce reprezintă de transferul de date. De asemenea, dacă discul nu este disponibil, o cerere de I/O strebuie să aştepte într-o coadă de cereri până când discul devine disponibil. [20]

O parte din algoritmii de planificare a accesului la disc suportați de mediul de simulare DiskSim sunt:

1. **First Come First Served (FCFS)** – este cel mai simplu algoritm de planificare a accesului la disc. În acest algoritm cererile sunt prelucrate în aceeaşi ordine în care sunt primite. Această metodă are avantajul de a fi corectă în sensul formal, dar se comportă într-o manieră similară cu planificarea aleatoare deoarece cererile I/O vin într-un mod aleator şi cel mai probabil nu vor accesa pistele de pe suprafaţa discului în ordinea potrivită. Algoritmul FCFS mai este cunoscut și sub numele de “First In First Out” (FIFO), “Run To Completion” sau “Run Until Done”.
2. **SCAN (Elevator)** – FCFS este singurul algoritm de planificare a acessului la datele de pe disc ce nu lasă cereri neprelucrate până la golirea cozii. Cu alte cuvinte, apare o problemă legată de prioritate: pot sosi mereu noi cereri care vor fi alese înainte de o cerere existentă. În cazul algoritmului SCAN, braţul se deplasează în ambele direcţii, satisfăcând toate

cererile, până când ajunge la ultima pistă într-o direcţie sau până când nu mai există cereri în acea direcţie. După ce ajunge la un capăt porneşte în sens opus. Este simplu de observat faptul că algoritmul SCAN favorizează pistele din centru deoarece braţul trece prin regiunea de mijloc mai des decât trece prin extremităţile discului.

3. **Circular SCAN (C-SCAN)** – restrânge scanarea într-o singură direcţie fiind similar cu algoritmul SCAN. Diferența este că atunci când ultima pistă a fost parcursă într-o anumită direcţie, braţul se întoarce la capătul opus al discului şi scanarea se reia de la început. Acest algoritm reduce întârzierea maximă cauzată de cererile noi.,

4. **Shortest Seek Time First (SSTF)** - selectează cererile I/O care necesită ca braţul să se depalseze cât mai puțin de la poziția actuală. Acest algoritm dovedește o performanță mul mai ridicată decât FCFS deşi o serie de decizii optime, fiecare la câte un pas nu garantează o soluţie optimă per ansamblu.

5. **VSCAN** - reprezintă o combinație între algoritmul SSTF (Shortest Seek Time) şi algoritmul SCAN (Elevator). VSCAN deserveşte cererile folosind algoritmul SSTF atunci când nu este necesară schimbarea direcţiei de deplasare a capului de citire/scriere. Furnizează un echilibru bun intre timpul mediu de răspuns şi rezistenţa la "înfometare" (fenomenul de întârziere a unor cereri).

6. **Shortest Positioning Time First (SPTF)** - Ca şi la SSTF şi SPTF apare problema fenomenului de "înfometare". Cererile care necesită un timp mai mare pentru poziţionare sunt amânate pe o perioadă lungă de timp. Pentru a reduce variaţia răspunsului în timp, cererile din coadă care sunt în aşteptare de o perioadă prea lungă de timp primesc prioritate. Prioritatea creşte o dată cu timpul petrecut de cerere în coadă, sau se poate stabili o limită de timp, după care cererilor în aşteptare să li se dea prioritatea maximă. O variantă a acestui algoritm este SATF (Shortest Access Time First). Acesta doreşte reducerea timpului de poziţionare împreună cu reducerea timpului de transfer.

7. **LOOK** – similar lui SCAN, capul circulă de-a lungul suprafeţei de disc satisfăcând cereri în direcţii alternantive. Capul de citire / scriere îşi schimbă direcţia de parcurgere dacă nu mai există cereri în acea direcţie, rezultând o reducere a numărului de cilindri parcurşi.

8. **Circular LOOK (C-LOOK)** - Algoritmul C-LOOK reprezintă o combinaţie între algoritmii C-SCAN şi LOOK. Capul de citire / scriere se mişcă la fel ca la algoritmul SCAN cu deosebirea că aici se mişcă doar până la ultima cerere în fiecare sens, după care inversează direcţia de deplasare imediat, iar pe calea de întoarcere , capul de citire al discului se opreşte. [20][22][23][24][25]

În continuare voi prezenta o listă cu algoritmii puși la dispoziție de mediu de simulare DiskSim 4.0

- FCFS
- ELEVATOR LBN

- CYCLE LBN
- SSTF LBN
- ELEVATOR
- CYCLE CYL
- SSTF CYL
- SPTF OPT
- SPCTF OPT
- SATF OPT
- WPTF OPT
- WPCTF OPT
- WATF OPT
- ASPTF-OPT

4.5 Specificațiile componentelor subsistemului I/O

- **Scheduling policy** – indică algoritmul de planificare folosit pentru a selecta următoarea cerere ce va fi îndeplinită

- **Cylinder mapping strategy** – precizează nivelul de cunoaștere a dispunerii datelor pe disc
  0 → înseamnă că singura informație disponibilă este cea cu privire la numărul blocurilor logice ale fiecărei cereri
  1 → indică faptul că planificatorul mai cunoaște și frontierele zonelor, numărul de sectoare / zonă și numărul de sectoare fizice / pistă pentru fiecare zonă
  2 → planificatorul are de asemenea acces la informațiile despre dispunerea sectoarelor și pistelor de rezervă din fiecare zonă
  3 → planificatorul are acces și la lista cu sectoarele și pistele "adormite";
  4 → planificatorul are acces la lista sectoarelor și pistelor remapate, asigurând astfel o mapare LBN corectă
  5 → planificatorul folosește numărul de cilindru dat împreună cu cererea, admițând experimente cu diverse mapări
  6 → planificatorul are acces numai la numărul mediu de sectoare/cilindru

- **Write initiation delay** – reprezintă o aproximare a întârzierilor de procesare ce au loc la scrierea cererilor, înaintea oricăror întârzieri de poziționare. Algoritmii de planificare folosesc această valoare care selectează cererile în funcție de întârzierea de poziționare

- **Read initiation delay** - reprezintă o aproximare a întârzierilor de procesare la citirea cererilor, ce au loc înaintea oricăror întârzieri de poziționare. Această valoare este folosită de algoritmii de planificare care selectează cererile în funcție de întârzierea de poziționare

- **Sequential stream scheme** – returnează o valoare întreagă, interpretată ca un câmp de date boolean. Aceasta fumizează informațiile despre concatenarea cererilor:

- bitul 0 → specifică dacă cererile de citire secvențiale sunt sau nu concatenate de planificator
- bitul 1 → specifică dacă cererile de scriere secvențiale sunt sau nu concatenate de planificator
- bitul 2 → specifică dacă cererile de citire sunt sau nu planificate împreună
- bitul 3 → specifică dacă cererile de scriere sunt sau nu planificate împreună
- bitul 4 → specifică dacă cererile de citire / scriere sunt planificate într-o ordine aleatoare, împreună, pentru a fi îndeplinite de disc

- **Maximum concat size** - lungimea maximă a unei cereri, rezultată din concatenarea a mai multe cereri secvențiale;

- **Overlapping request scheme** - specifică politica de planificare folosită pentru tratarea cererilor suprapuse (suprascrise):

1 → indică faptul că cererile care sunt complet suprascrise de alte cereri care au sosit după ele sunt incluse în acestea;
2 → adaugă la cele de mai sus și posibilitatea ca cererile sosite după suprascriere să fie adăugate, dacă sunt disponibile datele necesare;

- **Sequential stream diff maximum** - specifică distanța maximă dintre două cereri secvențiale dintr-un șir secvențial;

- **Scheduling timeout scheme** - acest parametru specifică schema de întârzieri a unei cozi multiple:

0 → indică faptul că cererile nu sunt mutate din coada de bază într-o coadă cu prioritate mai mare datorită întârzierilor excesive din cauza parcurgerii cozilor;
1 → cererile din coada de bază, care au depășit un anume timp de așteptare în coadă sunt mutate în una din cele două cozi de mare prioritate (coada pentru cererile foarte vechi sau cea pentru cererile eu prioritate ridicată);
2 → cererile din coada de bază care au așteptat cel puțin jumătate din timpul limită sunt mutate în coada pentru cereri vechi, căpătând o prioritate puțin mai mare.

4.6 Parametrii pentru organizarea datelor în matrici de discuri

Parametrii utilizaţi pentru organizarea datelor în matrici de discuri sunt configuraţi în blocurile „logorg” şi sunt următorii:

- **Addresing mode** – specifică cum este adresată organizarea logică a datelor. Array indică faptul că există un singur dispozitiv logic pentru întreaga organizarea logică. Parts indică faptul că dispozitivele de stocare back-end sunt tratate ca şi cum nu au nici o organizaţie logică, şi cererile sunt re-mapate în mod corespunzător.
- **Distribution scheme** – aceasta arată schema de distribuţie de date (care este necesară la sistemul de redundanţă). Asis indică faptul că nu apare nicio re-mapare. Striped indică faptul că datele sunt împărţite la dispozitivele din matrice. Random indică faptul că un disc aleator este selectat pentru fiecare cerere. Ideal indică faptul că o distribuţie ideală de date ar trebui să fie simulata prin alocarea cererilor de disc folosind algoritmul round robin.

- **Redundancy scheme** - aceasta precizează sistemul de redundanţă. Noredun (RAID0) indică faptul că nu exista nici o redundanţă. Shadowed (RAID1) indică faptul că una sau mai multe copii ale fiecărui disc de date sunt menţinute. Parity_disk indică faptul că un singur disc de paritate este menţinut pentru a proteja datele din celelalte discuri ale matricii. Parity_rotated (RAID5) indică faptul că blocurile de paritate sunt împărţite de-a lungul tuturor discurilor din matrice.
- **Devices** – lista cu numele dispozitivelor care vor face parte din organizaţie.
- **Stripe unit** – specifică dimensiunea unităţii de întreţesere.
- **Number of copies** – specifică numărul de exemplare ale fiecărui disc de date dacă organizarea logică prezintă redundanţa Shadowed. Altfel , parametrul este ignorat.
- **Copy choice on read** – specifică politica folosită pentru selectarea unui disc dintr-un set de discuri Shadowed care se va ocupa de cererile de citire sosite din moment ce oricare dintre ele poate face acest lucru. Acest parametru este ignorat dacă redundanţa Shadowed nu este selectată.
- **RMW vs. reconstruct** - precizează punctul de întrerupere în selectarea actualizărilor de paritate ca raport al discurilor de date care sunt actualizate. În cazul în care numărul de discuri actualizate de cererile de scriere este mai mic decât punctul de întrerupere, atunci blocul "vechi" de date, blocul "vechi" de paritate, şi blocul "nou" de date sunt folosite pentru a calcula paritatea nouă.
- **Parity stripe unit** – specifică dimensiunea unităţii de întreţesere pentru sistemul de redundanţă Parity_Rotated. Nu este obligatoriu să fie egală cu „stripe unit” , dar una trebuie să fie multiplul celeilalte.
- **Parity rotation type** – specifică cum este rotită paritatea de-a lungul discurilor din matrice:
  1 →simetric la stânga,
  2 → asimetric la stânga,
  3 → asimetric la dreapta,
  4 → simetric la dreapta. [19][18]

CAPITOLUL 5

Simulatorul de arhitecturi RAID - DiskSim. Rezultate experimentale

5.1 Simulatorul pentru arhitecturi RAID

Pentru partea practică a acestei lucrări a fost realizată o aplicație JAVA ce incorporează simulatorul DiskSim 4.0 și un generator de workload-uri. Aplicația poate fi rulată atât pe sistemele de operare Windows cât și pe platformele Linux.

Aplicația realizează comunicarea între utilizator, generatorul de workload-uri și simulatorul DiskSim prin intermediul unei interfețe grafice ce va fi prezentată în subcapitolul 5.1.1. Generatorul de workload-uri crează fișiere speciale ce sunt necesare simulatorului DiskSim pentru generarea rezultatelor.

5.1.1 Interfața principală

Interfața principală a aplicației este împărțită în 2 tab-uri. Primul tab este dedicat generatorului de workload-uri și tipului de disc folosit. Cele două tipuri de disk ce pot fi folosite sunt: Seagate Cheetah 146GB 15000RPM și Quantum Tornado 9GB 10000RPM.

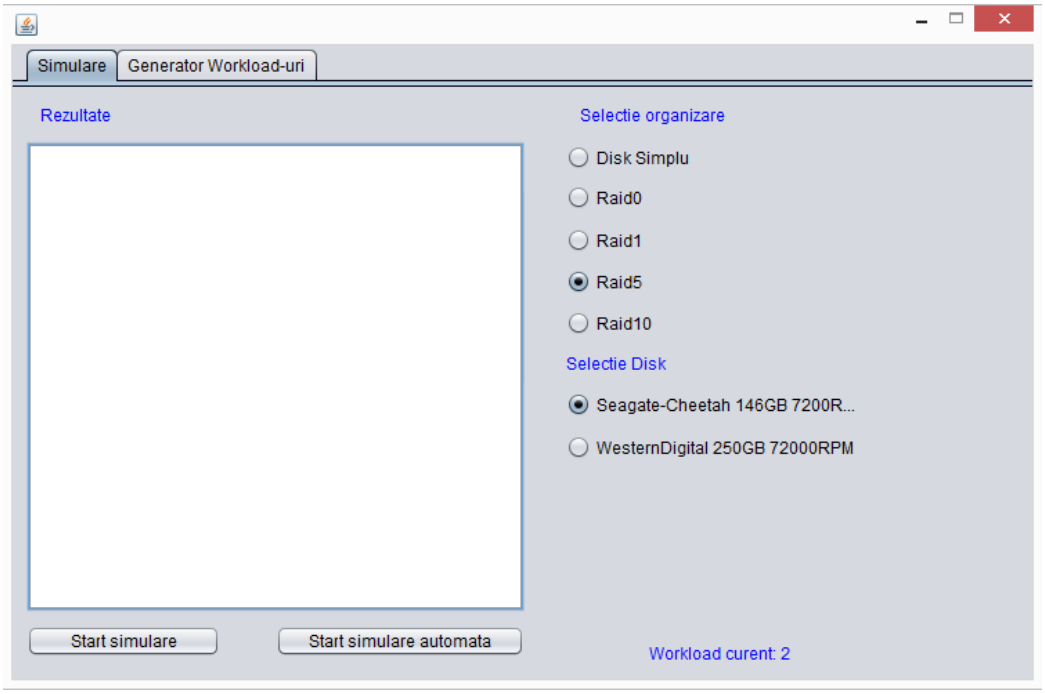

Figura 5.1
Interfața grafică a aplicației

Generatorul de workload-uri generează workload-uri în folder-ul “simulationfiles/workloads” sub numele “trace…..”. Programul nu poate selecta în vederea realizării unei simulări un workload din cele anterioare, ci îl folosește doar pe cel în curs. O dată ce s-a generat un workload nou, cel vechi nu mai poate fi folosit. În partea de jos a ecranului este afișat workload-ul curent.

Generatorul de Workload-uri permite configurarea anumitor parametrii ai workload-ului cum ar fi:

- Numărul de cereri
- Timpul între cereri
- Tipul de distribuție a cererilor

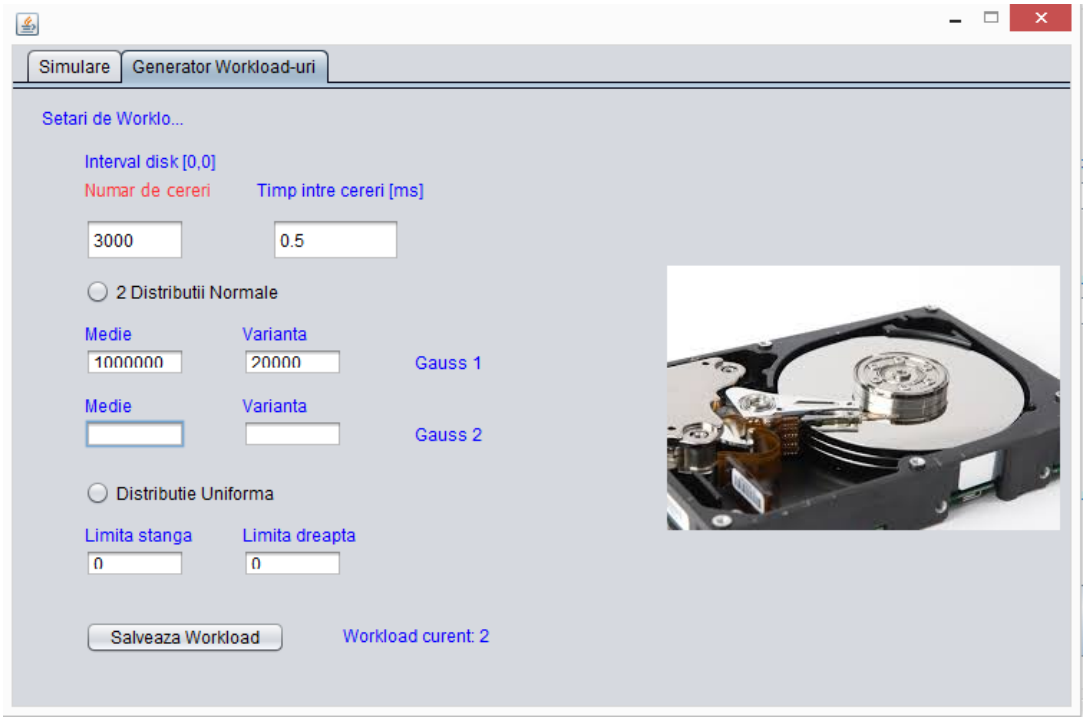

Fig. 5.2
Interfața aplicației – Generatorul de Workload-uri

Parametrii configurabili ai distribuției gaussiene sunt media și varianța acesteia. Atunci când se selectează și o a doua gaussiană, se utilizează 2 limite, una superioară și una inferioară pentru poziționarea distribuției pe disk. Cea de-a doua gaussiană va avea aceași varianță ca și prima.

Cele două gaussiene alternează în funcție de valoarea unei variabile aleatoare cu distribuție uniformă ce ia valori între [0, 1]. Dacă variabila este mai mică decât 0.5 atunci se alege prima gaussiană, altfel se alege cea de-a doua. Parametrii de control ai distribuției uniforme sunt capetele intervalului în care se vor poziționa cererile pe disk.

În partea superioară a căsuței există un câmp dedicat, scris cu font roșu, ce specifică intervalul în care cererile pe disk pot fi poziționate Unitatea de măsură este blocul de date și are 512 bytes.

Pentru a genera workload-ul final este nevoie ca butonul “Generează Workload” din josul căsuței să fie apăsat. Fișierul cu workload-uri va conține cereri generate sub un format standard.

```
0.0     0       31647817                1       1
0.2     0       33053507                1       1
0.4     0       30280558                1       1
0.6     0       27439933                1       1
0.8     0       30723408                1       1
1.0     0       97262470                1       1
1.2     0       30623897                1       1
1.4     0       98859013                1       1
1.6     0       32150787                1       1
1.8     1       30157532                        1
2.0     0       31348364                1       1
2.2     0       32708195                1       1
2.4     0       33375934                1       1
2.6     1       98744585                        1
2.8     0       102166903               1       1
3.0     0       29738271                1       1
3.2     0       102355882               1       1
3.4     0       104354412               1       1
3.6     1       98560747                        1
3.8     0       98355988                1       1
4.0     0       102848224               1       1
4.2     3       95017627                        1
4.4     0       98648855                1       1
4.6     0       29836755                1       1
4.8     0       33356153                1       1
5.0     0       103526302               1       1
```

Fig. 5.3
Formatul fișierului de workload

- Prima coloană reprezintă timpul la care sosește cererea față de momentul 0.
- A 2-a coloană reprezintă numărul organizării de disk-uri care se simulează
- A 3-a coloană indică numărul blocului accesat
- A 4-a coloană indică numărul de blocuri accesate în acea etapă
- Ultima coloană reprezintă tipul cererii. 1 pentru citire și 0 pentru scriere

Cadrul din mijloc este destinat selecției arhitecturii de discuri ce dorim a fi folosită și rulării efective a simulării. Putem alege între un Disk Simplu sau o matrice RAID 0, RAID 1, RAID 5 sau RAID 10. Dacă vom alege o matrice RAID 0 aceasta vine cu un număr presetat de 4 discuri. RAID 1 este presetat pe 2 discuri, RAID 5 are 5 discuri, iar RAID 10 are 4 discuri.

A doua setare ce poate fi făcută este legată de algoritmul de planificare a accesului la disc. Pornind de la cei 27 de algoritmi implementați în DiskSim se poate alege dintr-o listă simplificată ce include algoritmii FCFS, ELEVATOR LBN, CYCLE LBN, SSTF LBN sau SPTF OPT.

După ce am ales parametrii simulării putem face simularea apăsând butonul Start Simulare. În acest fel, folosind workload-ul generat se va genera o simulare, iar rezultatele vor fi afișate în consolă. La ieșire simulatorul va returna o serie de paramtrii. Aceșia sunt: “IOdriver Response time average” , “IOdriver Requests per second” , “IOdriver Number of reads” , “IOdriver Number of writes” ,

“IOdriver Request size average”. Aceste date vor fi salvate in fisierul “Rezultate.out” din folder-ul “simulationfiles”.

Cel de-a treia funcție a aplicației poate fi folosită independent de celelalte două și realizează o simulare automată. Simularea va fi realizată, pe rând, pentru fiecare tip de arhitectură și pentru fiecare tip de algoritm de planificare a accesului la disc. Simularea va fi efectuată doar asupra tipului de disc selectat.

Simularea automată va crea o serie de fișiere workload și apoi va apela simulatorul pentru fiecare tip de arhitectură și pentru fiecare tip de algoritm disponibil pe fiecare workload. Workload-urile reprezintă o înșiruire de cereri de citire sau scriere aplicate prin intermediul a două gaussiene. Una dintre ele este fixă, iar cealaltă se deplasează de la un capăt la celălalt al intervalului disk-ului folosit. Aceasta se deplasează cu un pas setat de parametrul “Rezoluție parcurgere”. La apăsarea butonului “Start Simulare” se vor crea toate workload-urile și apoi se va începe seria de simulări pe aceste workload-uri. Rezultatele sunt afișate în consolă sub forma unor perechi de numere (poziția blocului mediei celei de-a doua gaussiene și timpul mediu de răspuns la cereri măsurat în milisecunde). Aceste date se salvează și în fișierele “RezultateAuto…” din folder-ul “simulationfiles”.

5.1.2 Configurarea simulatorului

Integrarea simulatorului în programul JAVA, precum și realizarea legăturilor dintre simulator și interfața JAVA s-au realizat prin folosirea comenzii de consolă a simulatorului cu înlocuirea după caz a parametrilor:

Exemplu:

disksim raid1.1.parv trace1sim1.out ascii trace1.trace 0 "disc*" "scheduler: scheduling policy" 2

Pentru execuția simularilor, DiskSim folosește o serie de parametri specifici cum ar fi:

- Fișierul de parametri – ce are extensia .parv
- Fișierul cu specificațiile disk-ului – ce are extensia .diskspecs
- Fișierul de workload – ce are extensia .trace
- Alte fișiere de descriere a informațiilor (opționale sau nu)

Pentru că DiskSim nu permite creare de discuri noi în interiorul său s-a folosit utilitarul Dixtrac, special creat pentru a extrage parametrii de pe un disk existent. Utilitarul extrage toate trăsăturile unui disk real și le transformă în fișiere necesare simulatorului DiskSim. Acest program nu poate extrage date decât de pe un disk cu magistrală SCSI, pe o platformă Linux. După extragerea caracteristicilor discului, toate datele memorate pe acesta vor fi pierdute.

Comanda pentru apelarea utilitarului Dixtrac este “ ./new_disk.pl” și trebuie dată având setat ca director curent, folder-ul “/dixtrac”.

Pentru mai multe detalii privind implementarea simulatorului în program, se poate analiza codul sursă al simulatorului prezentat în cele ce urmează:

Aplicația este realizată din MainFrame.java , GenerateProcess.java , GenerateWorkload.java , ReadWithScanner.java și Rezultate.java.

S-au scris rând pe rând funcțiile pentru crearea workload-ului, setarea parametrilor workload-ului, scrierea datelor din fișirul workload etc.

```
public void genNorm(int nr, float mean, float var, float dist, int opt) {

    PrintWriter dst;
    try {
        Main.tracefile = "trace" + this.number + ".trace";
        String dstName = "simulationfiles/workloads/" + Main.tracefile;
        dst = new PrintWriter(new FileWriter(dstName));
    } catch (IOException e) {
        System.out.println("Cannot opent trace file!!!");
        return;
    }
    int i;
    String s;

    double val = 0;
    for (i = 0; i < nr; i++) {
        val = r.nextGaussian();
        if ((val >= -3) & (val <= 3)) {
            s = (float) (i * dist) + "\t0\t" + (int) (var * val + mean)
                    + "\t" + 1 + "\t" + this.rw(opt);
            dst.println(s);
        } else {
            i = i - 1;
        }
    }
    dst.close();
}
```

Fig 5.5
Crearea fișierului de workload

Programul crează un fișier cu extensia .trace ce trebuie să aibă un anumit format, în 5 coloane, prezentat în capitolul 5. O mostră dintr-un astfel de fișier se regăsește în figura 5.3

Generarea acestui fișier a fost realizată prin generarea unei valori random gaussiene. Prima funcție se ocupă de prima distribuție normală, fiind denumită intuitiv genNorm, în timp ce cea de-a 2a funcție denumită intuitiv **genNorm2** și reprezentată în figura **5.6a** se ocupă de cea de-a 2a distribuție normală.

A 3-a funcție denumită **genUnif** și prezentată în figura **5.6b** se ocupă de modelarea distribuției uniforme.

Aceste funcții sunt folosite pentru crearea workload-ului în funcție de dorințele utilizatorului. Secțiunile de cod au fost legate cu interfața grafică folosindu-se mediu de dezvoltare NetBeans.

În figura **5.7** sunt prezentate layar-ele interfeței grafice.

Este bine de știut faptul că aplicația în sine nu face simulările propriu zise ci apelează simulatorul de discuri DiskSim 4.0 prin intermediul interfeței grafice. Aplicația generează însă workload-urile.

```
    public void genNorm2(int nr, float medie1, float varianta1, float mean2,
60          float var2, float dist, int opt) {
61      float mean, var;
62
63      PrintWriter dst;
64      try {
65          Main.tracefile = "trace" + this.number + ".trace";
66          String dstName = "simulationfiles/workloads/" + Main.tracefile;
67          dst = new PrintWriter(new FileWriter(dstName));
68      } catch (IOException e) {
69          System.out.println("Cannot opent trace file!!!");
70          return;
71      }
72      int i;
73      String s;
74
75      double val = 0;
76      for (i = 0; i < nr; i++) {
77          if (r.nextInt(2) == 0) {
78              mean = medie1;
79              var = varianta1;
80          } else {
81              mean = mean2;
82              var = var2;
83          }
84
85          val = r.nextGaussian();
86          if ((val >= -3) & (val <= 3)) {
87              s = (float) (i * dist) + "\t0\t" + (int) (var * val + mean)
88                      + "\t" + 1 + "\t" + this.rw(opt);
89              dst.println(s);
90          } else {
91              i = i - 1;
92          }
93      }
94      dst.close();
95  }
```

Fig. 5.6a
genNorm2 – cod sursă

```
95      public void genUnif(int nr, int a, int b, float dist, int opt) {
96
97          PrintWriter dst, dst2;
98          try {
99              String dstName = "simulationfiles/workloads/trace" + this.number + ".trace";
100             String dstName2 = "simulationfiles/workloads/trace" + this.number + Main.soft + ".trace";
101             dst = new PrintWriter(new FileWriter(dstName));
102             dst2 = new PrintWriter(new FileWriter(dstName2));
103         } catch (IOException e) {
104             System.out.println("Cannot opent destination file!!!");
105             return;
106         }
107         int i, j;
108         String s, s2;
109
110         double val = 0;
111         for (i = 0; i < nr; i++) {
112             val = r.nextFloat() * (b - a) + a;
113             s = (float) (i * dist) + "\t0\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
114             dst.println(s);
115             s2 = (float) (i * dist) + "\t0\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
116             dst2.println(s2);
117             s2 = (float) (i * dist) + "\t1\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
118             dst2.println(s2);
119             s2 = (float) (i * dist) + "\t2\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
120             dst2.println(s2);
121             s2 = (float) (i * dist) + "\t3\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
122             dst2.println(s2);
123             s2 = (float) (i * dist) + "\t4\t" + (int) val + "\t" + 1 + "\t" + this.rw(opt);
124             dst2.println(s2);
125         }
126         dst.close();
127         dst2.close();
128     }
129
130 }
```

Fig. 5.6b
genUnif – cod sursă

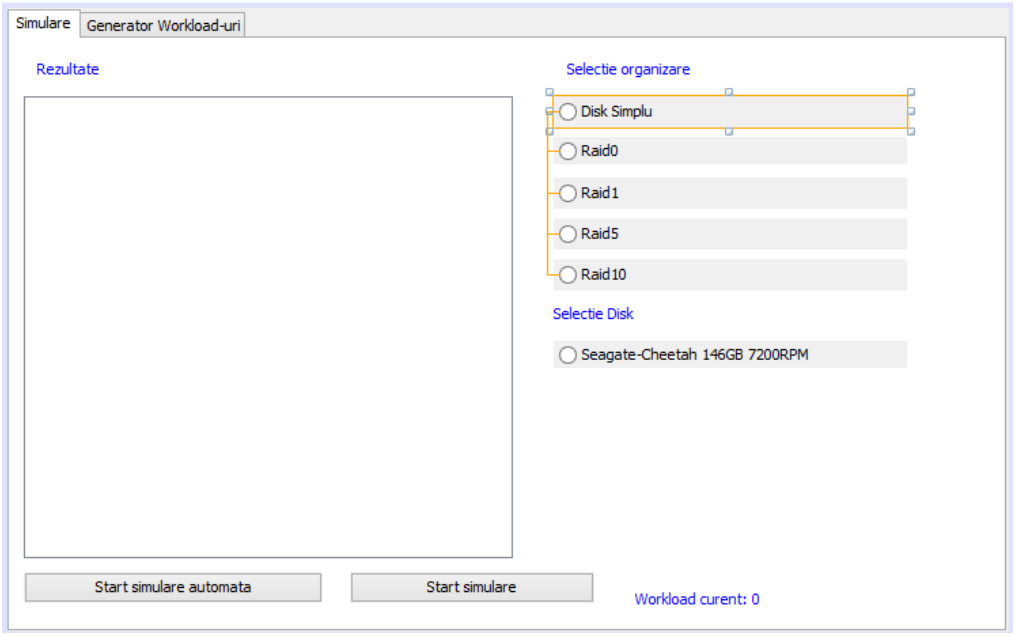

Fig. 5.7a
Legarea butoanelor din interfața grafică

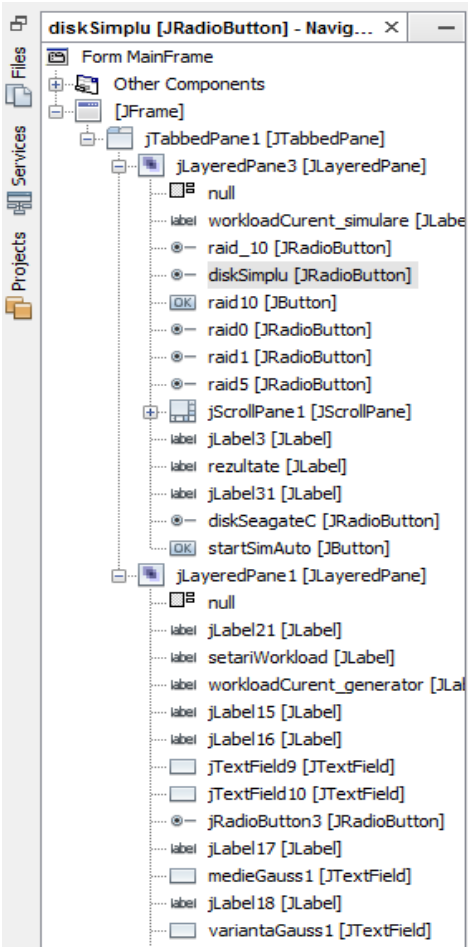

Fig. 5.7b

MainFrame.java începe cu blocul de control al workload-ului. Acesta citește parametrii setați și realizează simularea cerută.

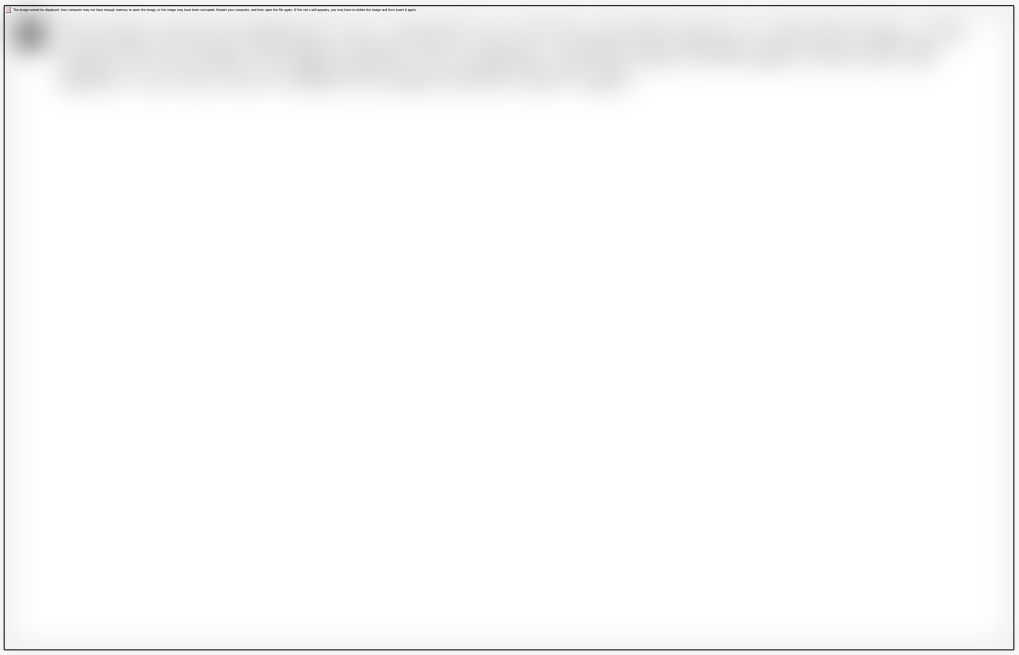

Fig. 5.8a

Fig. 5.8b

5.2 Rezultate experimentale

Pentru realizarea simulărilor s-a folosit o configurație ce specifică următorii parametrii:

- Număr de cereri – 3.000
- Timp între cereri – 0.5
- Medie – 100×10^6
- Varianță – 2×10^6
- Rezoluție parcurgere – 5×10^6

Folosind generatorul de workload-uri au fost realizate 50 de fișiere workload. Au fost generate atât sarcini doar de scriere sau citire cât și sarcini mixte, ce din punct de vedere matematic simulează evoluția unui proces de rebuilding.

Workload-urile ce simulează doar scrieri au coloana responsabilă formată strict din valoarea 0, iar cele responsabile doar cu citiri au coloana respectivă formată doar din valoarea 1.

Am constatat experimental că din punct de vedere matematic pentru a simula o procedură de rebuilding este nevoie ca (în cazul unei matrice RAID 5, spre exemplu, cu 5 discuri – în care cade un disc) după o secvență de 4 citiri consecutive (care simulează citirea de pe discurile rămase "în picioare") să avem o scriere, care să simuleze reconstrucția datelor abia culese pe discul ce este reconstruit. Desigur aceste procese pot fi intercalate cu citiri și / sau scrieri aleatoare ce simulează faptul că celelalte discuri sunt folosite în timp ce se reconstruiește discul căzut.

Pentru a simula cât mai multe situații s-a folosit funcția de simulare automată a simulatorului. Au fost generate simulări care să includă toate combinațiile posibile între algoritmi de planificare a accesului la disc, tipul de matrice RAID folosită și cele 50 de workload-uri.

Rezultatele se găsesc pe CD-ul atașat lucrării, precum și în anexă. Folosind aceste date s-au construit graficele de performanță ale structurilor RAID. Aceste grafice au fost realizate folosind mediul de dezvoltare Microsoft Excel.

În continuare vor fi prezentate o serie de grafice care ilustrează rezultatele experimentelor realizate asupra matricelor RAID. Acestea demonstrează anumite caracteristici prezentate în partea teoretică a lucrării. Aceste grafice sunt grupate în funcție de:

- **Latența rotațională** – acest parametru reprezintă timpul necesar suprafeței (unde se face operația de citire sau scriere) să se rotească până la poziția capului de citire
- **Timpul de acces** – acest parametru reprezintă intervalul de timp dintre momentul în care discul primește o cerere și momentul soluționării cererii respective împreună cu returnarea rezultatului.
- **Timpul de căutare** – acest parametru reprezintă unul din cele 3 tipuri de întârzieri associate cu citirea sau scrierea datelor pe un disc. Pentru a putea scrie sau citi (de) pe un disc, capul de citire trebuie mutat fizic la poziția cu pricina. Acest procedeu se numește "căutare". Distanța variază în funcție de cât de departe de destinație este capul de citire și de timpul necesar fiecărei instrucțiuni de citire sau scriere. Se va considera timpul mediu.
- **Timpul de transfer** – acest parametru reprezintă timpul necesar unui disc pentru a transfera un singur bloc de 512bytes pe magistrală.

Rezultatele generate de simulator au fost prelucrate cu ajutorul unui manager de text, realizat în limbaj JAVA. Acesta analizează fișierele .outv și extrage parametrii necesari aranjându-i într-un tabel pentru a se putea construe graficul de performanță.

Pentru algoritmul de acces FCFS s-au obținut următoarele rezultate:

Din punct de vedere al Latenței Rotaționale:

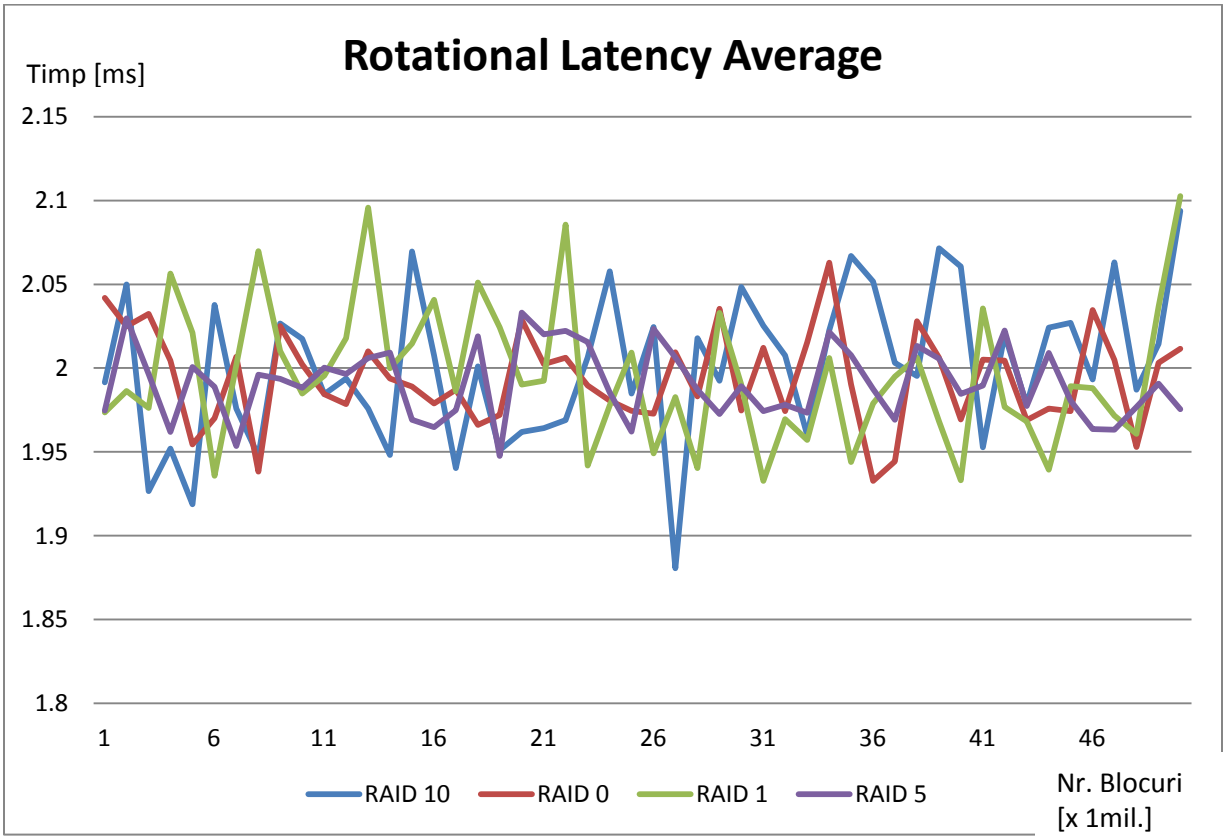

Fig. 5.10

Rotational Latency Average

Observând graficul de mai sus, unde sunt suprapuse cele 4 rezultate grafice aferente sistemelor RAID 0, RAID 1, RAID 5 și RAID 10 putem observa faptul că latențele rotaționale ale acestora nu variază în limite foarte mari, dar sunt totuși diferite. RAID 10 înregistrează minime ale latenței rotaționale ce se situează undeva în jurul valorii de 1.88ms, în timp ce valorile cele mai mari sunt atinse de către RAID 1 (2ms).

Din punct de vedere al timpului de căutare:

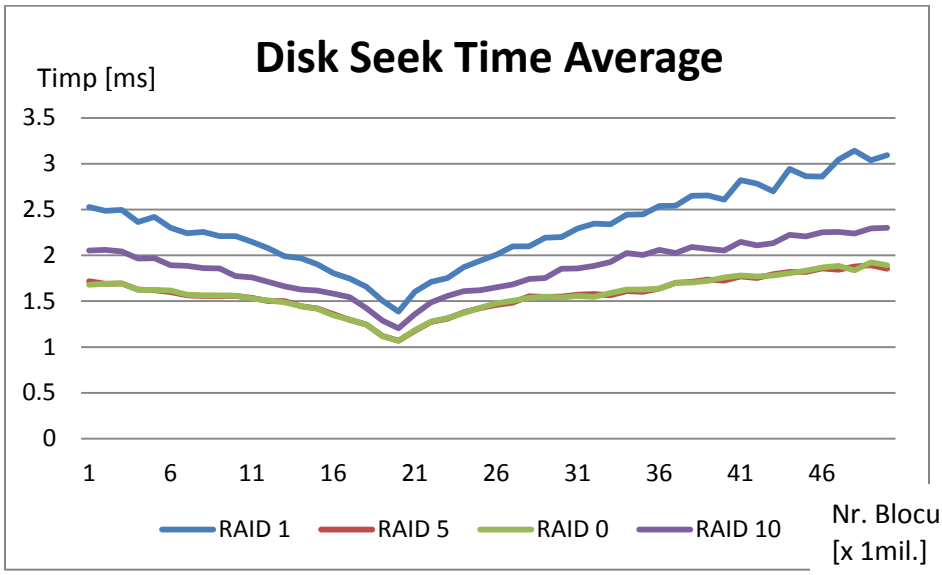

Fig. 5.11a

Disk Seek Time Average

Timpii de căutare pe disk variază în funcție de sistemul RAID folosit conform prezentărilor teoretice realizate în capitolele anterioare. Se poate observa că timpul cel mai scurt de acces la disk este deținut de RAID 0 și RAID 5, în timp ce timpul cel mai lent îi aparține lui RAID 1.

Din punct de vedere al Timpului Mediu de Răspuns

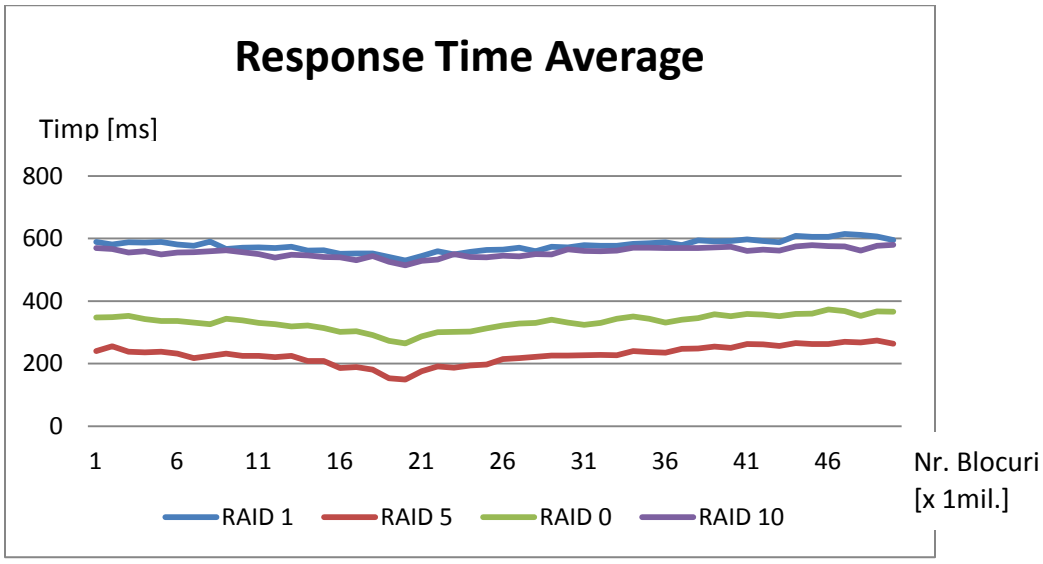

Fig. 5.11b

Response Time Average

Din punct de vedere al timpului mediu de acces la disk

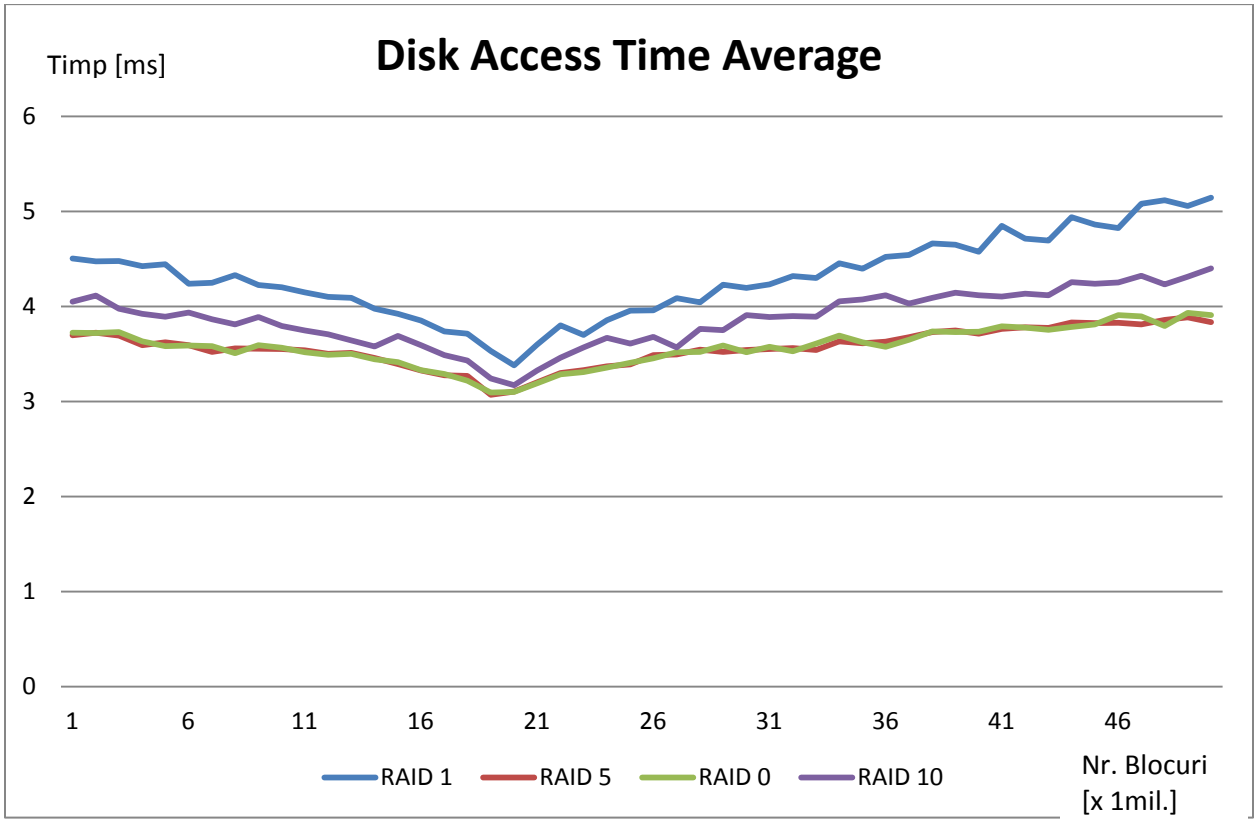

Fig. 5.12

Disk Access Time Average

Timpul de acces la disk variază proporțional între cele 4 tipuri de RAID analizate. Conform datelor prezentate teoretic, se observă că RAID 0 și RAID 5 împart locul întâi la capitolul timp de accesare al discului.

Pe mijlocul graficului, undeva aproape de mijlocul numărului de blocuri de date se poate observa un minim al timpului măsurat. Acesta se produce (în toate graficele) datorită faptului că seek-ul este făcut aleator, fiind favorizate pistele de pe mijlocul discului. Căutarea pe diverse piste pleacă din orice pistă și ajunge pe orice pistă.

Pentru a evidenția rezultatele diferite obținute în cazul folosirii unui alt algoritm de acces la disk în continuare sunt prezentate aceleași analize asupra discului, dar folosind algoritmul SSTF LBN.

Pentru algoritmul de acces la disc SSTF LBN s-au obținut rezultatele:

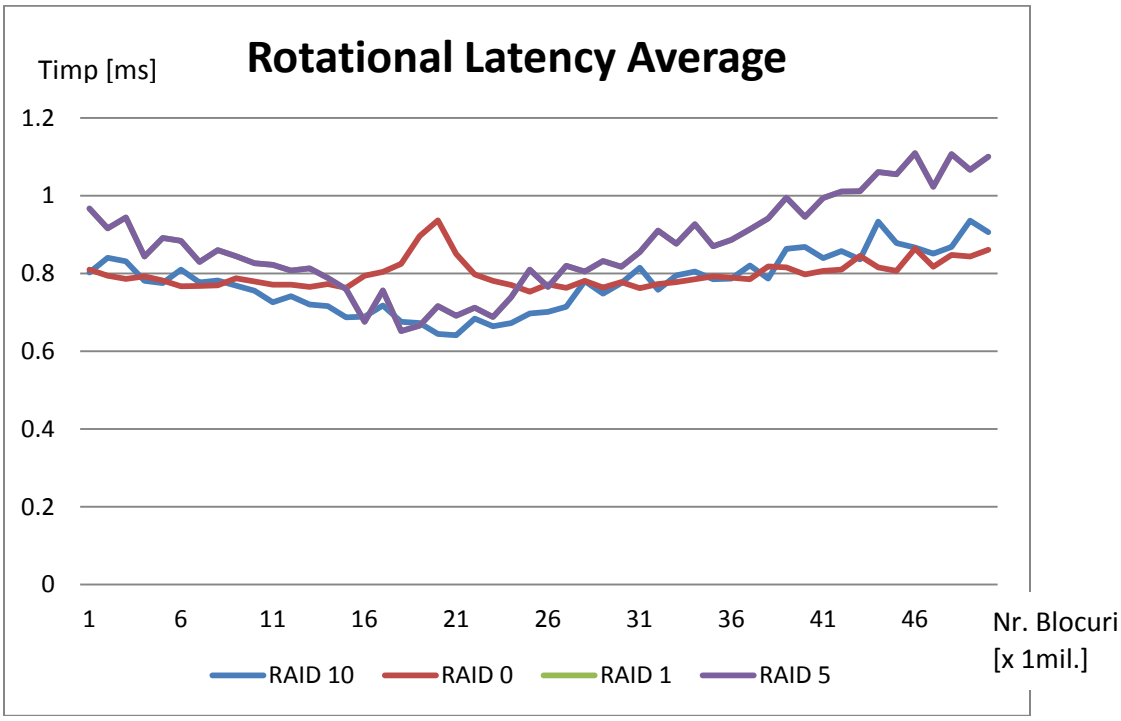

Fig. 5.13a
Rotational Latency Average

Pentru algoritmul de acces la disc SSTF LBN s-au obținut rezultatele:

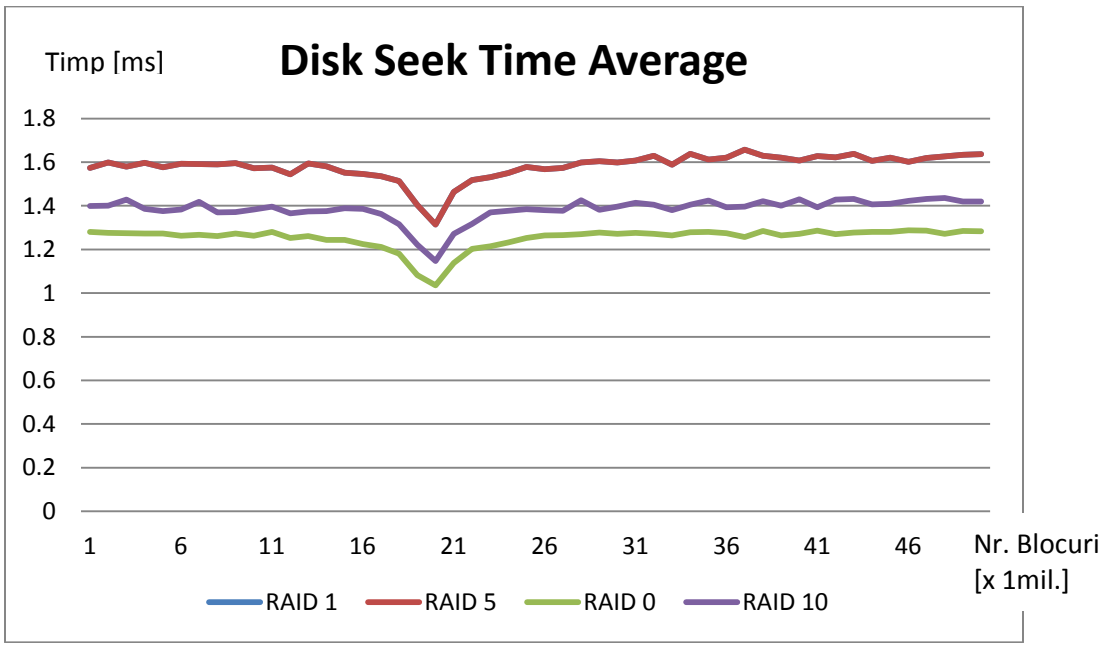

Fig. 5.13b

Disk Seek Time Average

Pentru algoritmul de acces la disc SSTF LBN s-au obținut rezultatele:

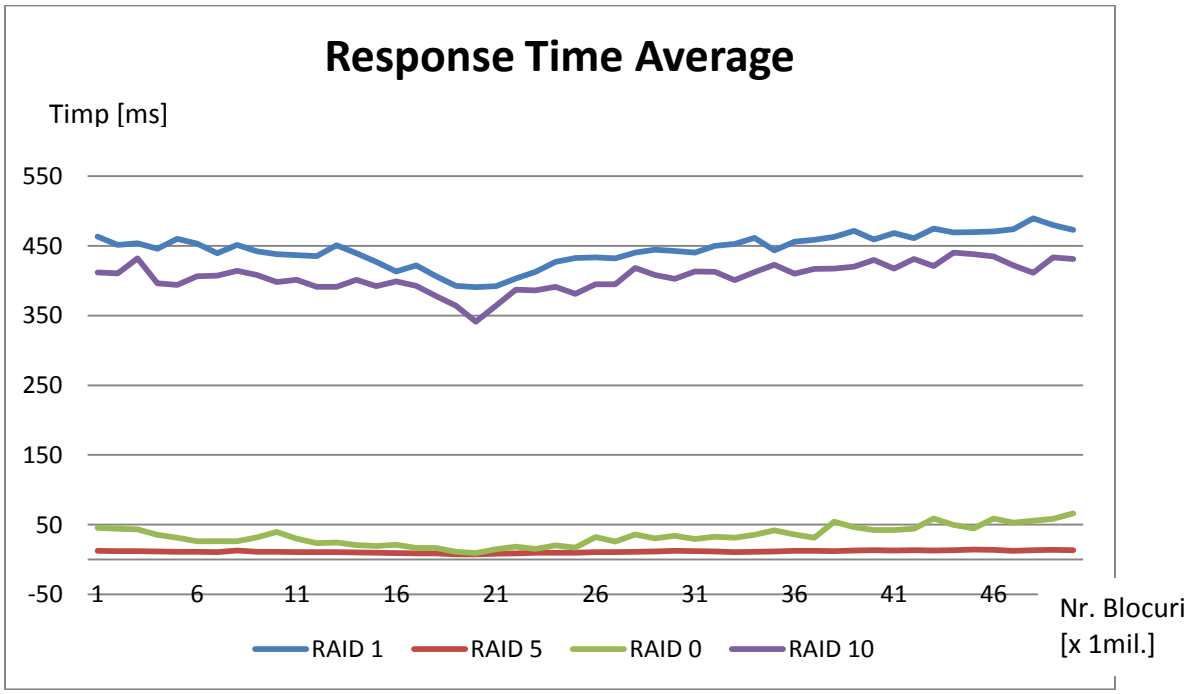

Fig. 5.14a
Response Time Average

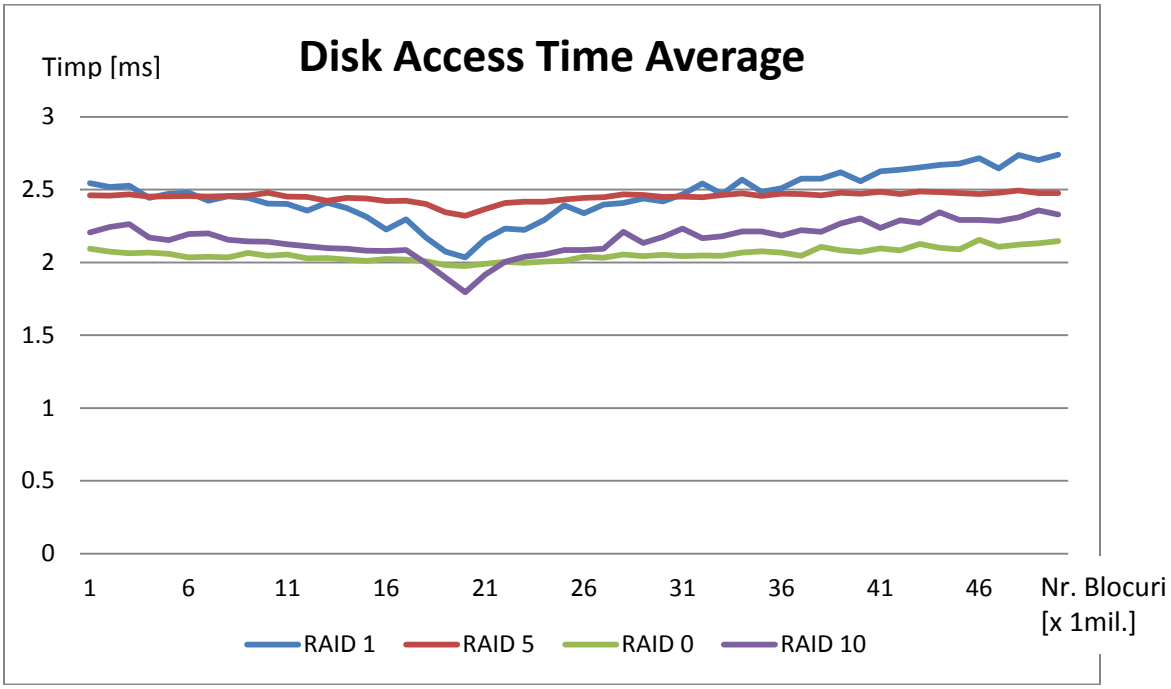

Fig. 5.14b
Disk Acces Time Average

5.3 Concluzii

Rularea simulărilor prezentate anterior ne-au permis conturarea următoarelor concluzii:

În primul rând, aruncând o privire de ansamblu asupra graficelor observăm faptul că RAID 5 oferă performanțe mai bune decât RAID 0, RAID 1 și RAID 1+0 la aproape toate capitolele. În cazul Latenței Rotaționale se observă rezultate ce diferă nesemnificativ, iar în cazul Timpului de Căutare se observă performanțe similare între RAID 5 și RAID 0.

Performanțele unui RAID privind timpii de căutare nu pot fi îmbunătățite prin legarea HDD-urilor într-un RAID anume. Singura metodă viabilă de imbunătățire a timpului de căutare rămâne procurarea unui harddisk ce oferă nativ un timp de căutare mai bun, cum ar fi spre exemplu SSD-urile. (*Solid State Disk*) Nici o matrice mecanică nu va reuși să atingă performanțele obținute de un SSD. Performanţa citirilor în RAID 5 creşte cu numărul de discuri, deoarece cu cât sunt mai multe discuri, cu atât sunt mai multe capete de citire/scriere iar matricile RAID 5 au capacitatea de a citi simultan de pe toate discurile.

Este adevărat totuși că un RAID poate îmbunătăți ușor timpul de căutare atunci când are un număr mare de "*queue depths*" (QD). Un utilizator de rând poate atinge un QD de 3-10 cereri, în timp ce un server are peste 20. Totodată RAID mărește de asemenea și latența întrucât procesarea cererilor necesită un *overhead* mai mare. Pentru home useri, în esență, timpul de căutare va scădea în urma folosirii unui sistem RAID față de un singur disk mai rapid.

O alternativă ar putea consta în folosirea unei curse scurte (short – stroking). În acest fel se forțează harddiskul să folosească doar partea exterioară a platanului, unde viteza liniară este mai mare.

Din punct de vedere al timpului mediu de acces la disk putem afirma că se evidențiază din nou RAID 5 și RAID 0. Aceste două tipuri de RAID prezintă timpi de acces similari, cu o constantă fixă adăugată în funcție de latența dorită a rotației platanului.

Timpul mediu de acces reprezintă suma dintre timpul mediu de căutare (variabil) și latența rotațională medie (stabilită de viteza platanului).

Timpul de căutare este bazat pe durata medie necesară deplasării între piste aleatoare, iar latența rotațională reprezintă timpul necesar platanului pentru a efectua o jumătate de rotație. O jumătate de rotație pentru că acesta este timpul mediu necesar pentru a aduce bit-ul căutat sub capul de citire.

RAID 0 stochează sau citește o parte din informații de pe un harddrive și cealaltă parte de pe celălalt harddrive beneficiind de viteză sporită tocmai datorită faptului că are nevoie să acceseze doar jumătate din date per harddrive fizic. Performanțele vor crește o dată cu numărul de harddisk-uri implicate, întrucât secțiunile de date sunt distribuite egal pe ambele HDD-uri ale matricei, oferind fiecărui HDD mai puțin de făcut pentru fiecare bucată de dată în parte.

Un alt factor ce influențează destul de mult performanțele discurilor este algoritmul de acces la disc ales. Primul set de rezultate este bazat pe algoritmul FCFS care din cauza simplității sale furnizează rezultatele cele mai slabe. Al doilea set de rezultate este obținut folosind algoritmul SSTF LBN observându-se rezultate mai bune.[26]

CAPITOLUL 6

Analizatorul de arhitecturi RAID - Iometer. Rezultate experimentale

6.1 Prezentarea mediului de dezvoltare Iometer

Pentru a putea observa comportarea sistemelor RAID într-un mediu real de funcționare am efectuat o testare a unei matrice RAID 5 cu discuri conectate printr-un controller Smart Array 5300, folosind mediul de simulare Iometer.

Iometer este un tool de măsurare și caracterizare a subsistemelor I/O ce fac parte din sisteme *single* sau *cluster* ce poate măsura performanțele sub un load controlat. Acesta este în același timp un generator de workload-uri și un tool de măsurare, putând genera operații de I/O asupra sistemului și să măsoare răspunsul acestuia. Iometer poate fi folosit pentru a emula load-ul de pe un disk sau de pe o rețea sau poate fi folosit pentru a genera load-uri I/O complet sintetice.

Programul poate măsura:

- Performanța disk-urilor și a controllerelor de rețea
- Lățimea de bandă și capabilitățile de latență ale magistralelor
- Performanțele magistralelor share-uite
- Performanțele hard-drive-urilor la nivel de sistem

6.2 Interfața programului

Iometer este compus din două programe separate: Iometer și Dynamo.

Iometer este programul de control. Folosind interfața grafică a Iometer-ului se poate configura workload-ul, se pot seta parametrii de operare și se pot începe sau se pot opri testele. Iometer îi transmite programului Dynamo ce să facă, colectează datele și apoi exportează într-un fișier de ieșire rezultatele obținute.

Dynamo este generatorul de workload-uri. Acesta nu are o interfață grafică. La comanda programului Iometer acesta efectuează operații I/O și înregistrează informațiile cu privire la performanțe, apoi le returnează programului Iometer. Generatorul de workload-uri Dynamo este de tip multithread și fiecare copie a acestuia, ce poate fi rulată în paralel, poate simula workload-ul mai multor programe. Fiecare copie a programului Dynamo care lucrează poartă numele de *manager*, iar fiecare thread al fiecărei copii Dynamo este numită generic *worker*.

Interfața grafică a programului Iometer este prezentată în figura următoare

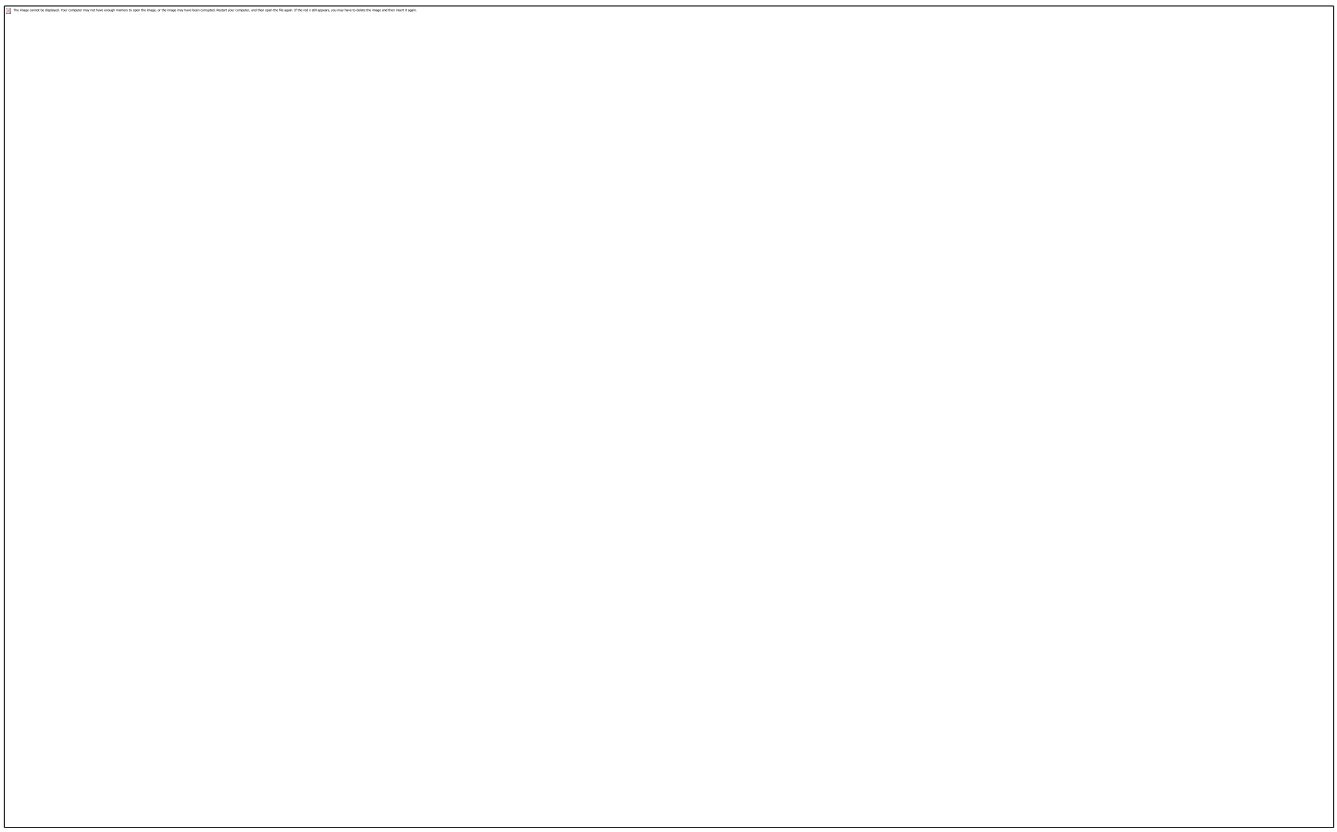

Fig. 6.2
Iometer - Interfața grafică

Opțiunile de configurare ale acestui program constă în stabilirea numărului de sectoare, stabilirea sectorului de unde va începe testarea, stabilirea modului de scriere sau citire asupra discului și împărțirea acestor sarcini pe thread-uri.

Rezultatele obținute sunt afișate parțial în interfața grafică și exportate în totalitate într-un fișier excel.

6.3 Rezultate experimentale

În urma testelor efectuate s-a obținut un set de rezultate ce sunt anexate lucrării, pe CD, în formatul original, excel.

S-au efectuat mai multe teste asupra matricei RAID 5, începând cu generarea de workload-uri pe un singur thread asupra configurației aflate în regim normal de funcționare și asupra configurației ce ulterior a fost introdusă în Recovery Mode. S-a continuat prin testarea configurației folosind 4 thread-uri și respectiv 1 thread atunci când aceasta efectuează Rebuilding în condiții de prio Normal, Low și High (pentru Rebuilding).

Primul test efectuat s-a realizat folosind un singur worker. Ca și specificații de acces la disk s-a folosit opțiunea globală "All-in-One" care înglobează un mix realizat automat între toate opțiunile posibile oferite de Iometer. Al 2-lea test efectuat s-a realizat introducând diskul în Recovery Mode.

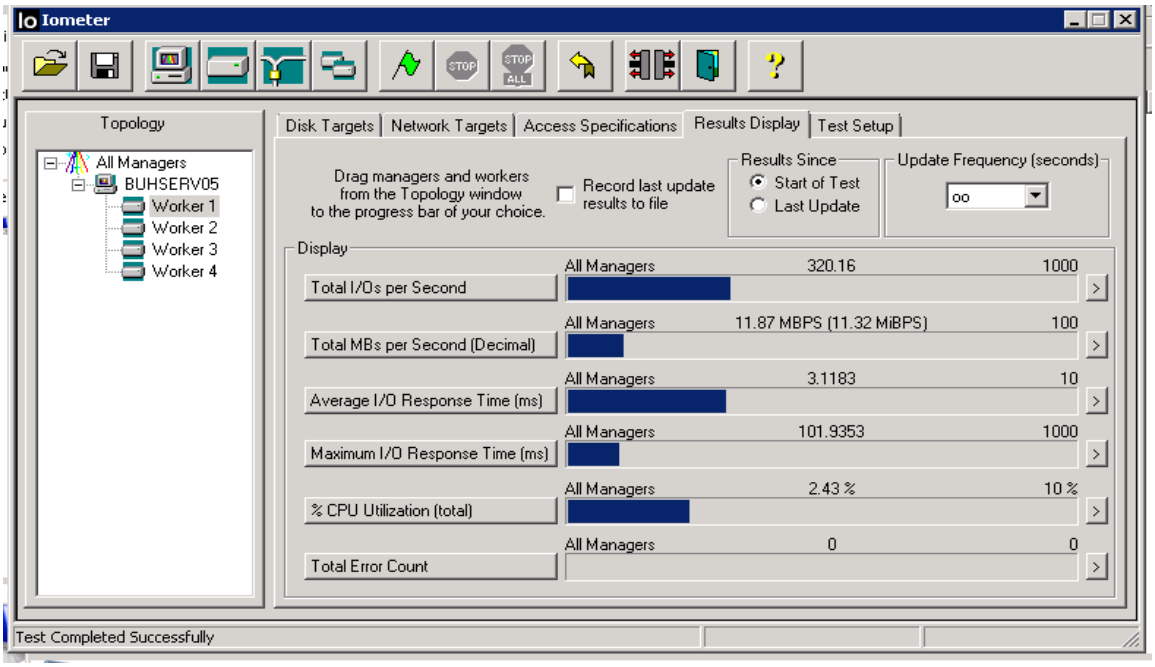

Fig. 6.3a
Analiza 1 - RAID 5 – 1 worker – Algoritm All-in-One

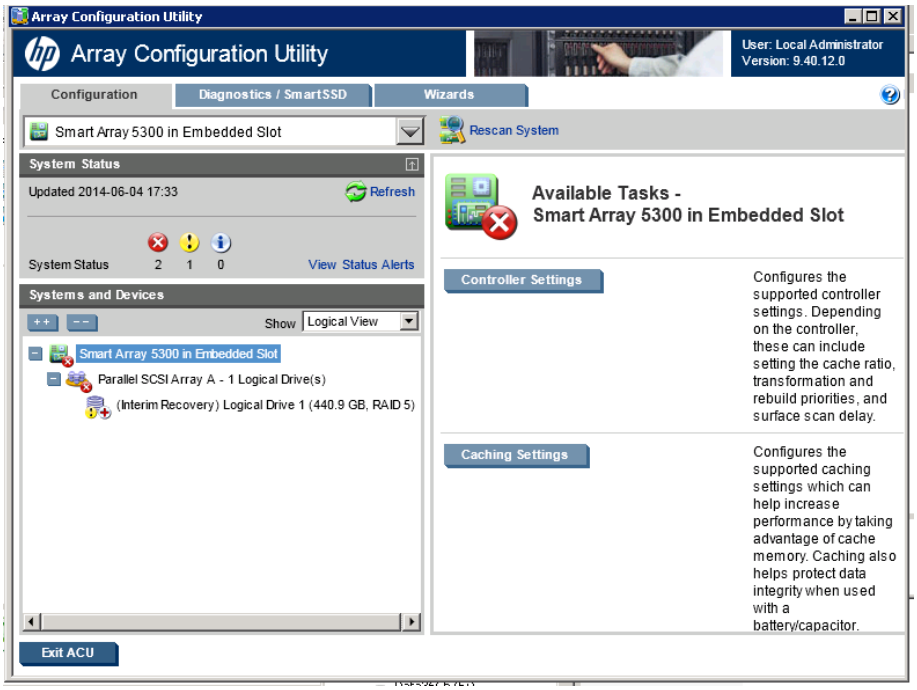

Fig. 6.3b
Analiza 2 – RAID 5 – 1 worker – Recovery Mode

Pentru a nu distruge un disk am simulat producerea unui eveniment nefericit asupra acestuia prin scoatera manuală, reconectarea la un alt sistem și ștergerea manuală a datelor de pe acesta. Ulterior pentru a o induce disk-ul în starea de Rebuilding, acesta a fost introdus la loc în sistemul inițial.

Rezultatele obținute pot fi observate în figura de mai jos.

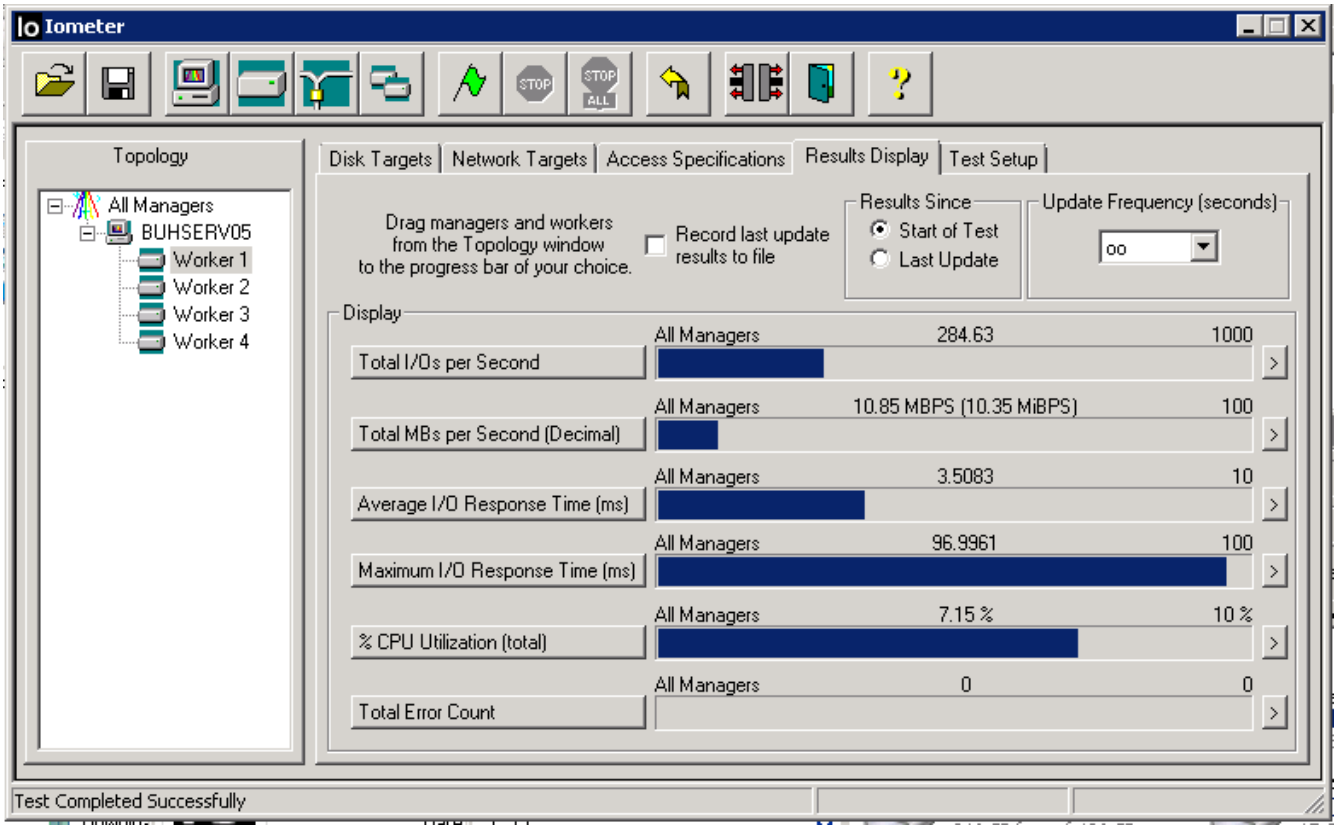

Fig. 6.4
Analiza 2 – RAID 5 – 1 worker – Recovery Mode

Se observă o încărcare mult mai puternică a procesorului (7.15% față de 2.43%) precum și o scădere a vitezei medii totale. (10.85MBPS atunci când diskul este în Recovery Mode față de 11.87MBPS atunci când diskul funcționează în parametrii normali). Numărul de operații de I/O pe secundă este și el în scădere, de la 320.16 în condiții normale de funcționare până la 284.63 atunci când diskul se află în Recovery Mode. Timpul maxim de răspuns la operațiile de I/O scade de la 101.9353 ms în cazul funcționării în parametrii normali la 96.9961ms în cazul funcționării în regim de Recovery Mode. De asemenea atunci când disk-ul intră în Recovery Mode, timpul de răspuns mediu la operațiile I/O crește de la 3.1183ms la 3.5083ms.

Testarea performanțelor în condiții de rebuilding a produs următoarele rezultate:

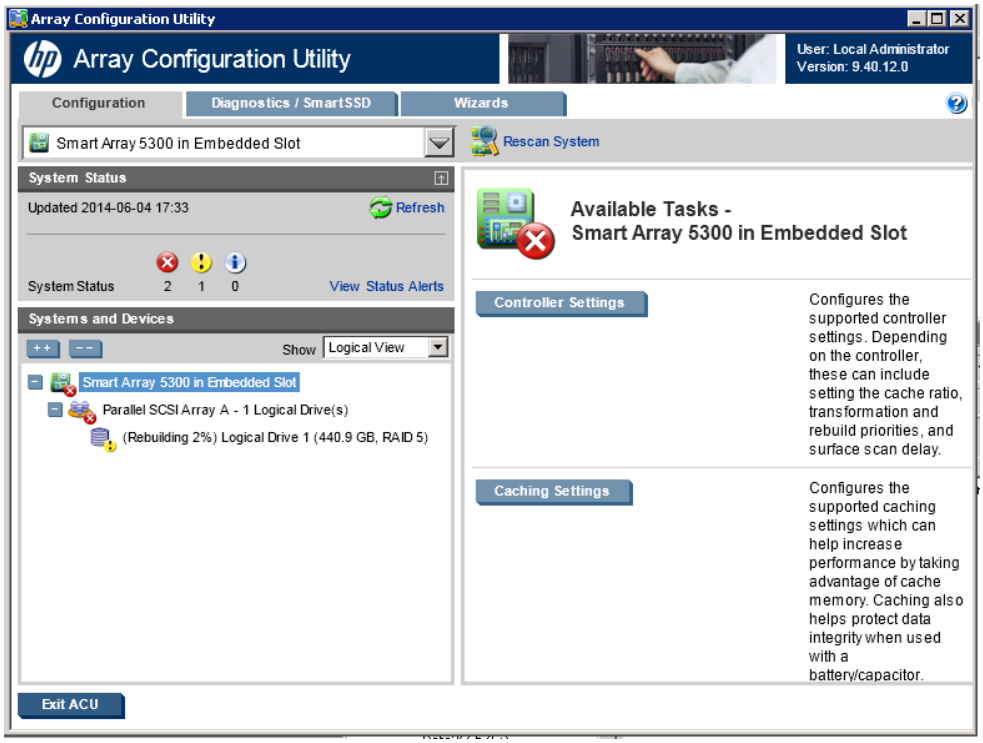

Fig. 6.5a
RAID 5 în procesul de rebuild

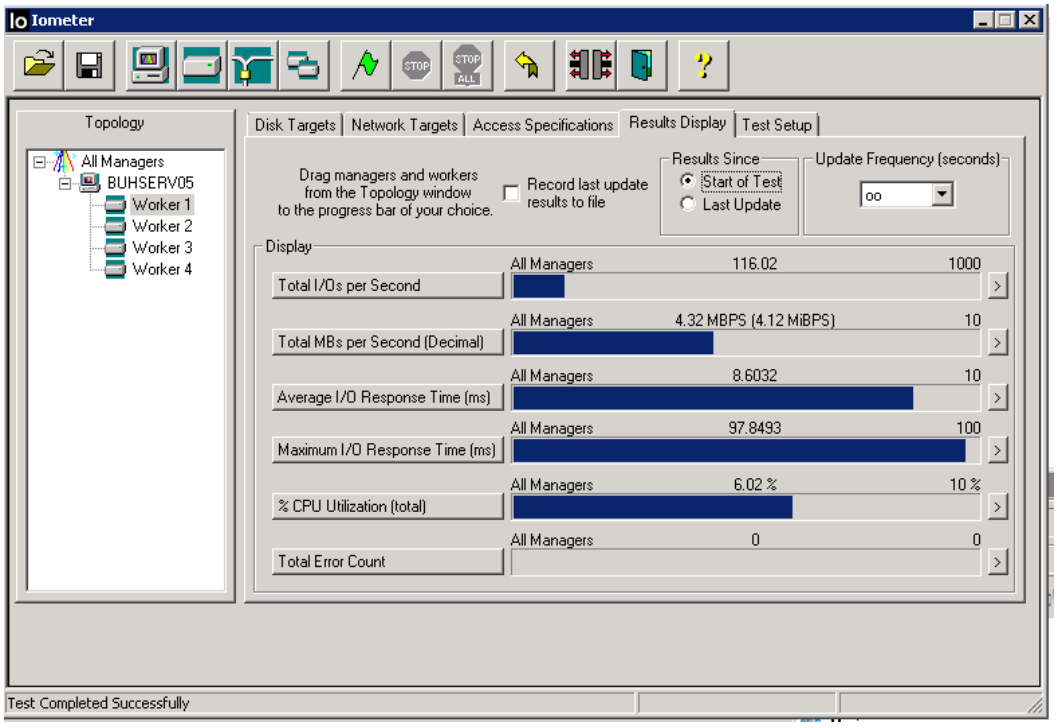

Fig. 6.5b
Rezultate experimentale – RAID 5 – Rebuilding – 1 worker – Normal Prio

Față de rezultatele obținute în urma rulării în parametrii normali, se observă din nou o creștere semenificativă la nivelul de încărcare al procesorului. (6.02% în timpul rebuilding-ului față de 2.43% în condiții normale de rulare)

Întrucât în timpul efectuării operației de rebuild sistemul este mai încărcat se observă o creștere considerabilă a timpului mediu de răspuns la operațiile de I/O. Creșterea are loc de la 3.1183ms pentru sistemul folosit în parametrii normali de funcționare până la 8.6032ms pentru folosirea acestuia în timpul operației de rebuild.

Totodată se observă că traficul făcut scade de la 11.87 MBPS în condiții normale de utilizare până la 4.32 MBPS atunci când are loc procesul de rebuild.

De asemenea numărul total de IOps scade de la 320.16 în condiții normale de utilizare la 116.02 în timpul realizării rebuilding-ului.

Paralelizând cererile furnizate de către generatorul de workload-uri dar păstrând prioritatea acordată rebuilding-ului observăm câteva schimbări semnificative. Aceste schimbări sunt extrem de importante pentru performanța procesului de rebuilding, evident existând anumite soluții de rebuilding mai bune decât altele.

Paralelizarea workloadurilor aduce un plus la nivelul tuturor parametrilor măsurați. Numărul total de IOps crește de la 116.02 (pentru 1 worker) la 346.38 (pentru 4workeri), viteza de transfer crește de la 4.32 MBPS la 12.83 MBPS, timpul mediu de răspuns crește și el din nefericire de la 8.6032ms la 11.4990 însă nu este o creștere la fel de semnificativă ca cea a tuturor parametrilor "pozitivi". Toate aceste creșteri semnificative ale performanțelor vin cu un cost ce se observă la nivelul procesorului. Încărcarea acestuia crește considerent de la 6.02% până la 16.05%.

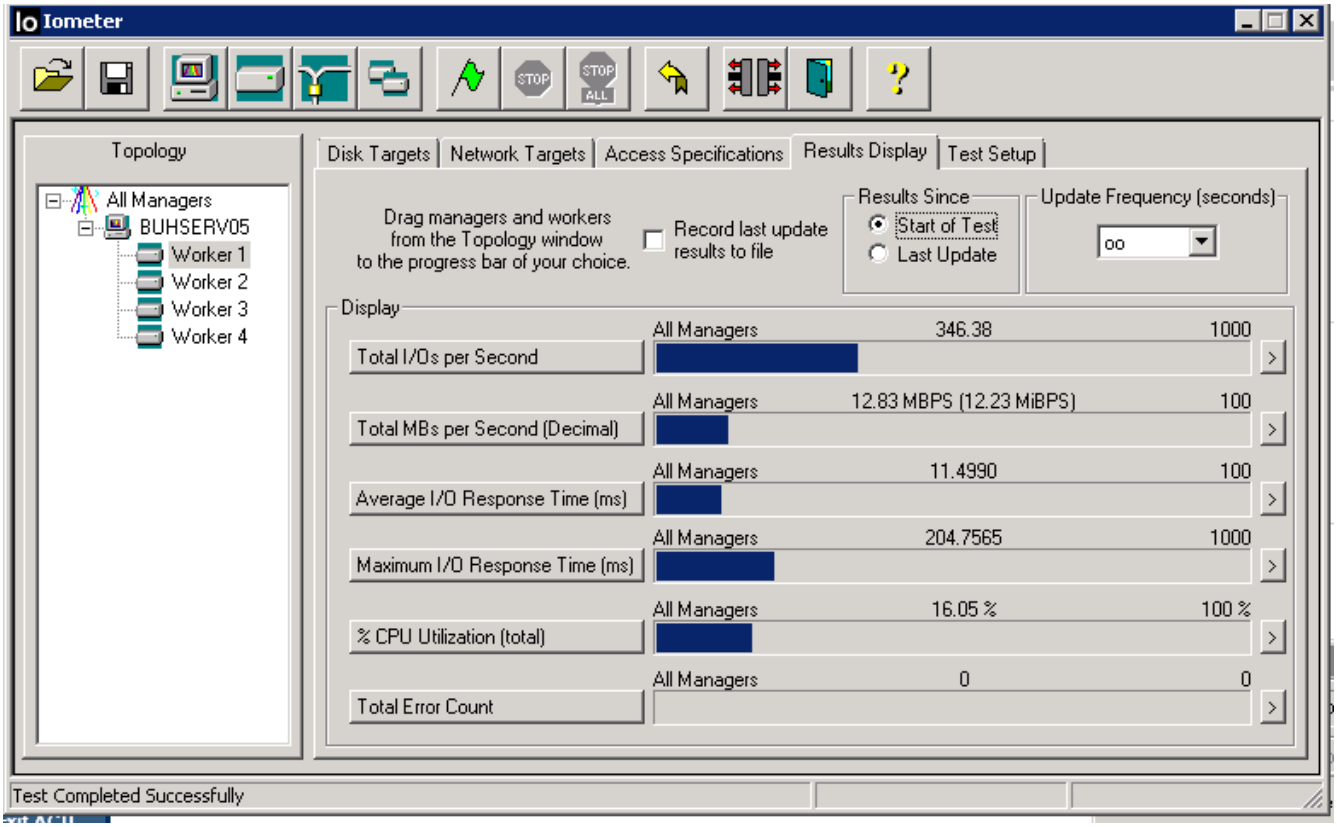

Fig. 6.6
Rezultate experimentale – RAID 5 – Rebuilding – 4 workeri

Următorul test a fost făcut folosind aceiași parametrii (1 worker) și în timpul aceluiași proces de rebuilding dar prioritatea procesului de reconstrucție a fost trecută pe „Low”.

Se observă cum load-ul procesorului scade de la 16.05% la 3.98% dar o dată cu acesta și restul indicilor de performanță raportează valori mai scăzute. Acest rezultat se datorează faptului că oferind o prioritate scăzută procesului de rebuilding, controllerul se poate concentra pe operațiile efectuate asupra discurilor rămase în stare de funcționare.

Această metodă este preferată dacă se urmărește folosirea discurilor rămase, dar nu este indicată atunci când scopul principal este de a realiza un rebuilding cât mai rapid.

Numărul total de IOps scade de la 346.38 (în cazul priorității normale și 4 workeri) până la 277.56 (în cazul priorității scăzute cu 1 worker). Rezultatul obținut este totuși mai bun decât cel rezultat pentru prioritatea normală cu 1 worker și anume 116.02 IOps. Totodată viteza de transfer scade de la 12.83 MBPS (în cazul priorității normale și 4 workeri) până la 10.65 MBPS. Scorul obținut este oricum mai bun decât cel pentru prioritatea normală cu 1 worker și anume 4.32 MBPS.

La nivelul timpilor de răspuns medii respectiv maximi la operațiile I/O se observă o coborâre a valorilor de la 11.4990ms la 3.5992ms și respectiv de la 204.7565ms la 169.8599ms. Aceste rezultate sunt bune dacă ne dorim lucrul cu discurile rămase, dar nu se apropie de calitatea celor rezultate în cazul procesului de rebuilding realizat cu prioritate normală și anume 8.6032ms pentru timpul mediu de răspuns și respectiv 97.8493ms pentru timpul maxim de răspuns.

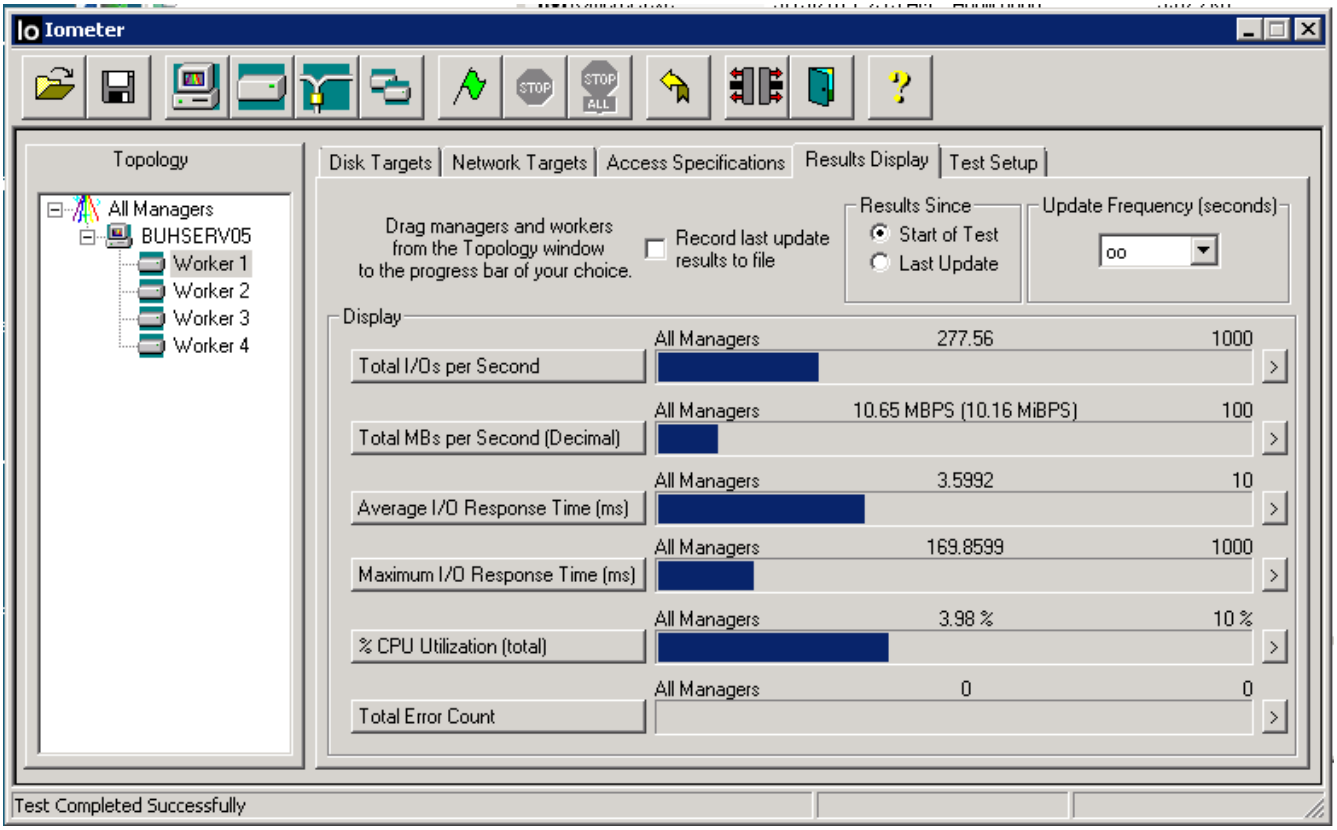

Fig.6.7
Rezultate experimentale – RAID 5 – Rebuilding – 1 worker – Low Prio

Ultimul experiment realizat folosește o aceași configurație asupra căreia s-a realizat rebuilding cu prioritate ridicată (High). Workloadul a fost realizat de către un singur worker.

Se observă cum întrucât toate resursele sunt canalizate pentrea realizarea cât mai rapidă a rebuildingului, numărul de IOps scade dramatic de la 277.56 (pentru Low Priority) și 346.38 (pentru Normal Priority) până la 81.53.

De asemenea transferul de date se face la viteze mult mai mici. Vechile performanțe atinse (10.65 MBPS – pentru Low Prio și respectiv 4.32 MBPS – pentru Normal Prio ; ambele cu 1 worker) depășesc cei 3.10 MBPS posibili în cazul High Prio cu 1 worker.

Încărcarea procesorului crește până la 5.30%.

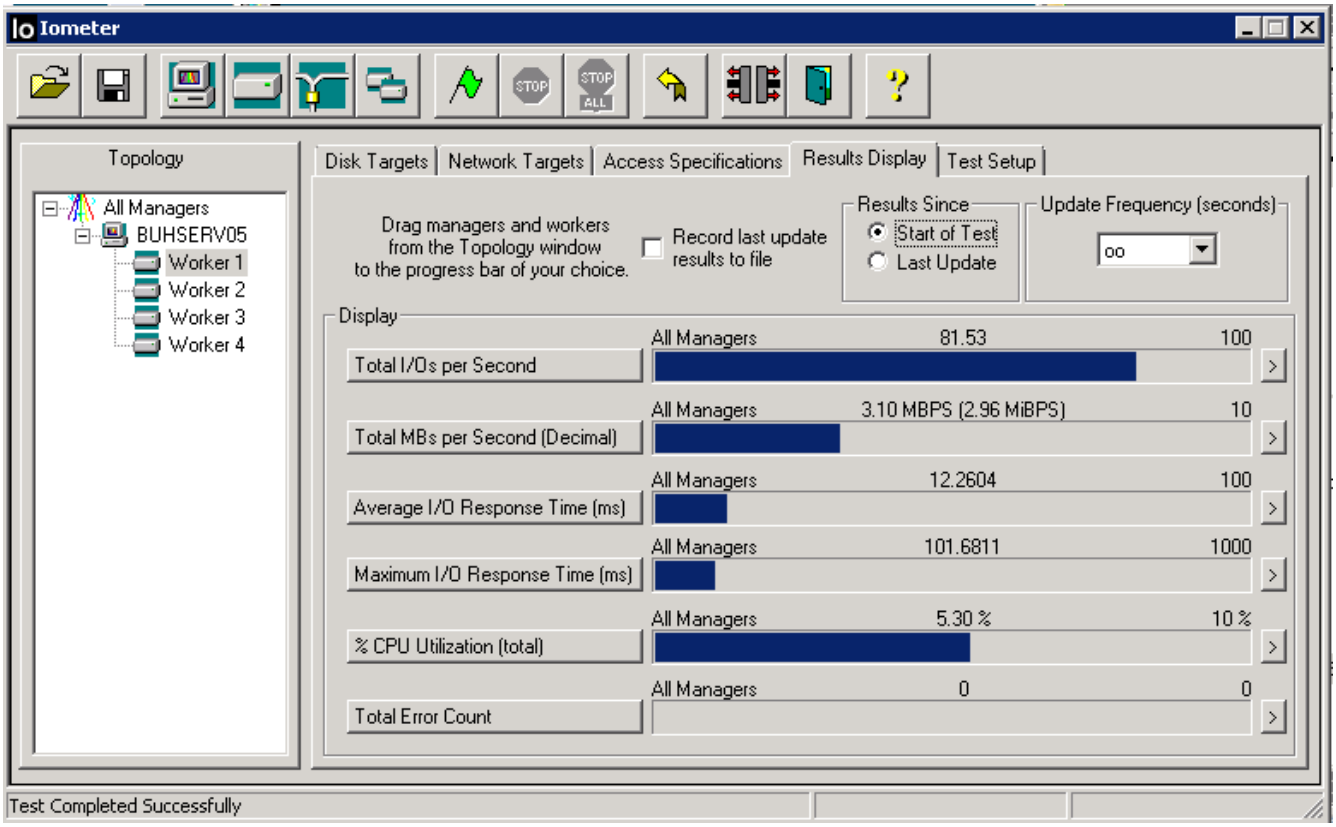

Fig. 6.8
Rezultate experimentale – RAID 5 – Rebuilding – 1 worker – High Prio

CAPITOLUL 7

Considerente economice

Utilizarea tehnologiilor RAID aduce o serie indiscutabilă de avantaje economice. Dezvoltarea sistemelor RAID nu a survenit doar ca un pas natural al evoluției tehnologiei ci și ca un răspuns la nevoile utilizatorilor.

În primul rând, sistemele RAID oferă siguranță și viteză. Cele mai multe RAID-uri se găsesc în mediu buisness și enterprise. Este evident că marile companii multinaționale își doresc viteză și redundanță din considerente economice. Chiar dacă folosindu-se discuri simple în configurații standard ar crește capacitățile de stocare, pierderile aduse în urma defectărilor discurilor ar depăși cu mult economiile făcute prin folosirea unor metode clasice de stocare și gestionare a datelor.

Un alt avantaj major adus de RAID este puterea de acces. Serverele ce folosesc sisteme RAID permit conectarea utilizatorilor din mai multe locații și surse, precum și stocarea datelor diferite ce nu au legătură una cu alta. În acest fel se economisește mult spațiu, pentru ca sistemele profesionale de stocare ocupă foarte mult spațiu și totodată din punct de vedere logistic, toate operațiile asupra echipamentelor se vor face în același loc. În acest fel se vor diminua costurile privind deplasarea tehnicienilor în locații îndepărtate una față de alta.

Un alt avantaj important îl reprezintă unificarea tehnologiilor. Folosind sisteme RAID cu același mod de funcționare, depanarea și servisarea acestor echipamente este mai ușor de făcut. Un lucru ce s-a evitat din totdeauna din raționamente economice a fost folosirea mai multor tehnologii în cadrul aceluiași proiect. În primul rând pentru că din cauza unei defecțiuni acest lucru ar putea conduce la costuri suplimentare privind mentenanța sistemelor și în al doilea rând pentru ca pregătirea personalului calificat și instruirea utilizatorilor ar fi mult prea dificil de realizat.

Creșterea productivității este un alt mare aport economic adus de folosirea tehnologiilor RAID. Pierderea datelor ar reprezenta o catastrofă economică pentru firme. Prin folosirea sistemelor RAID ce au redundanță protecția datelor este asigurată. În caz că unul din discurile matricei cedează, singurele costuri ce vor apărea vor fi legate de înlocuirea sau repararea componentei hardware defecte. Recuperarea datelor se va realiza în mod automat folosind funcțiile RAID nemaifiind nevoie de intervenția altor persoane. Astfel se economisesc atât timp cât și bani.

În concluzie, utilizarea tehnologiilor RAID este o necesitate datorită avantajelor economice pe care le aduce. Folosirea acestora este recomandată atât pentru beneficiile economice aduse cât și pentru cele tehnologice.

Bibliografie

1. [9] – “A Performance Evaluation of RAID Architectures” – Shenze Chen, *Member, IEEE,* and Don Towsley, *Fellow, IEEE*
2. [10] – „Data Allocation In Disk Arrays with Multiple RAID Levels” – Jun Xu, Dissertation Paper, New Jersey Institute of Technology, 2008
3. [11] – „A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation” – Liping Xiang and Yinlong Xu, University of Science and Technology of China – 2011
4. [12] – „SCAN: An Efficient Decoding Algorithm for RAID-6 Codes” – Jianqiang Luo, Lihao Xu – Wayne State University Detroit
5. [24] – “Rețele de Calculatoare, Ediția a 4-a” – Andrew S. Tanenbaum – Universitatea Vrijie, Amsterdam
6. [25] – „Operating Systems: Design and Implementation, 3rd edition” – Andrew S. Tanenbaum - Universitatea Vrijie, Amsterdam
7. [16] - "Performance analysis of disk arrays under failure" - R. R. Muntz and J. C. S. Lui.
8. [17] - "Performance analysis of RAIDS disk arrays with a vacationing server model" - A. Thomasian and J. Mcnon.
9. [18] – „The DiskSim Simulation Environment Version 4.0 Reference Manual” – John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger , 2008
10. [19] - „The DiskSim Simulation Environment Version 3.0 Reference Manual” – John S. Bucy, Gregory R. Ganger , 2003
11. [20] - “I/O Management and Disk Scheduling” CSc33200: Operating Systems, CS-CCNY, Fall 2003 Jinzhong Niu December 8, 2003

12. [22] – Structuri de Date și Algoritmi – Note de Curs - Sorin Zoican
13. [23]- Sisteme de Operare – Note de Curs – Conf. Dr. Ing. Ștefan Stăncescu

14. [1] – http://searchstorage.techtarget.com/feature/Learn-all-about-RAID-storage - Greg Shulz, 2008 – accesat la data 02.07.2014
15. [2] - http://en.wikipedia.org/wiki/RAID - 09.03.2014
16. [3] - http://ro.wikipedia.org/wiki/RAID - accesat la data 08.03.2014
17. [4] - http://www.revistait.ro/tehnologie/totul-despre-raid/ - accesat la data 01.04.2014
18. [5] - http://www.maxpedia.ro/educatie2/schema-de-alocare-a-fisierelor-pe-medii-de-stocare.php - accesat la data 11.04.2014
19. [6] - http://blogu.lu/kassak/schema-de-alocare-a-fisierelor-pe-medii-de-stocare - accesat la data 11.04.2014
20. [7] - http://www.acnc.com/raidedu/3 - accesat la data 22.06.2014
21. [8] - http://en.wikipedia.org/wiki/Standard_RAID_levels - accesat la data 18.05.2014
22. [13] - http://en.wikipedia.org/wiki/Nested_RAID_levels - accesat la data de 15.04.2014
23. [14] - http://www.freeraidrecovery.com/library/raid-rebuild.aspx - accesat la data 12.06.2014
24. [15] - http://techdepo.ro/recuperarea-datelor-de-pe-un-hard-disk-cu-probleme/# - accesat la data 14.06.2014
25. [21] - http://wiki.lug.ro/RAID - accesat la data de 8.05.2014
26. [26] - http://www.tomshardware.co.uk/forum/225521-32-should-raid-faster-raid-reads - accesat la data 22.06.2014

ANEXE

Anexa 1 – Managerul de text

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;

namespace Balauca
{
 /// <summary>
 /// Structura prin care se memoreaza valorile din fiecare fisier.
 /// </summary>
 public struct Items
 {
  public decimal? item1;
  public decimal? item2;
  public decimal? item3;
  public decimal? item4;
  public decimal? item5;
 }

 public partial class Form1 : Form
 {
  /// <summary>
  /// Cheia 1.
  /// </summary>
  public string item1 = "Rotational latency average";
  /// <summary>
  /// Cheia 2.
  /// </summary>
  public string item2 = "Transfer time average";
  /// <summary>
  /// Cheia 3.
  /// </summary>
  public string item3 = "Disk seek time average";
  /// <summary>
```

```
/// Cheia 4.
/// </summary>
public string item4 = "Response time average";
/// <summary>
/// Cheia 5.
/// </summary>
public string item5 = "Disk acces time average";

public Form1()
{
 InitializeComponent();
 writeToControls();
}

private void button1_Click(object sender, EventArgs e)
{
 //Citeste valorile de cautat.
 readFromControls();

 OpenFileDialog openDialog = new OpenFileDialog();
 openDialog.Multiselect = true;

 List<Items> items = new List<Items>();
 if (openDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
 {
  //Pentru feicare fisier deschis se cauta Cheile definite.
  foreach (string fileName in openDialog.FileNames)
  {
   String[] fileLines = File.ReadAllLines(fileName);
   Items item = new Items();

   //Se parcurg toate liniile fisierului.
   foreach (string line in fileLines)
   {
    if (!item.item1.HasValue && line.Contains(item1 + ": \t"))
    {
     int x = line.LastIndexOf("\t");
     item.item1 = decimal.Parse(line.Remove(0, x + 1), CultureInfo.InvariantCulture);
    }

    if (!item.item2.HasValue && line.Contains(item2 + ": \t"))
    {
     int x = line.LastIndexOf("\t");
     item.item2 = decimal.Parse(line.Remove(0, x + 1), CultureInfo.InvariantCulture);
    }

    if (!item.item3.HasValue && line.Contains(item3 + ": \t"))
    {
     int x = line.LastIndexOf("\t");
     item.item3 = decimal.Parse(line.Remove(0, x + 1), CultureInfo.InvariantCulture);
```

```
      }

      if (!item.item4.HasValue && line.Contains(item4 + ": \t"))
      {
        int x = line.LastIndexOf("\t");
        item.item4 = decimal.Parse(line.Remove(0, x + 1), CultureInfo.InvariantCulture);
      }

      if (!item.item5.HasValue && line.Contains(item5 + ": \t"))
      {
        int x = line.LastIndexOf("\t");
        item.item5 = decimal.Parse(line.Remove(0, x + 1), CultureInfo.InvariantCulture);
      }
    }

    items.Add(item);
  }
 }


 SaveFileDialog saveFile = new SaveFileDialog();
 saveFile.Filter = "Excel Worksheets|*.xls";
 if (saveFile.ShowDialog() == System.Windows.Forms.DialogResult.OK)
 {
  //Se realizeaza scrierea in fisierul Excel.
  WriteExcel(saveFile.FileName, items);
 }
}

/// <summary>
/// Creeaza un fisier Ecel cu datele obtinute.
/// </summary>
/// <param name="path">Calea si numele fisierului.</param>
/// <param name="items">Lista cu parametrii de scris.</param>
public void WriteExcel(string path, List<Items> items)
{
 Excel.Application xlApp;
 Excel.Workbook xlWorkBook;
 Excel.Worksheet xlWorkSheet;
 object misValue = System.Reflection.Missing.Value;

 xlApp = new Excel.Application();
 xlWorkBook = xlApp.Workbooks.Add(misValue);

 xlWorkSheet = (Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
 int count = 1;
 foreach (Items item in items)
 {
  xlWorkSheet.Cells[count, 1] = item.item1;
  xlWorkSheet.Cells[count, 2] = item.item2;
```

```
      xlWorkSheet.Cells[count, 3] = item.item3;
      xlWorkSheet.Cells[count, 4] = item.item4;
      xlWorkSheet.Cells[count, 5] = item.item5;
      count++;
    }

    xlWorkBook.SaveAs(path, Excel.XlFileFormat.xlWorkbookNormal, misValue, misValue, misValue, misValue,
Excel.XlSaveAsAccessMode.xlExclusive, misValue, misValue, misValue, misValue, misValue);
    xlWorkBook.Close(true, misValue, misValue);
    xlApp.Quit();

    System.Runtime.InteropServices.Marshal.FinalReleaseComObject(xlWorkBook);
    System.Runtime.InteropServices.Marshal.FinalReleaseComObject(xlWorkSheet);
    System.Runtime.InteropServices.Marshal.FinalReleaseComObject(xlApp);
    GC.Collect();
    MessageBox.Show("Excel file created!");
  }

  /// <summary>
  /// Citeste valorile de cautat in controale.
  /// </summary>
  private void readFromControls()
  {
    item1 = textBox1.Text;
    item2 = textBox2.Text;
    item3 = textBox3.Text;
    item4 = textBox4.Text;
    item5 = textBox5.Text;
  }

  /// <summary>
  /// Scrie valorile in controale.
  /// </summary>
  private void writeToControls()
  {
    textBox1.Text = item1;
    textBox2.Text = item2;
    textBox3.Text = item3;
    textBox4.Text = item4;
    textBox5.Text = item5;
  }

 }
}
```