UNIVERSITATEA POLITEHNICA TIMIŞOARA
Facultatea de Automatică şi Calculatoare

TEZĂ DE DOCTORAT

Creşterea fiabilităţii la transmiterea şi stocarea informaţiei

Conducător ştiinţific
Prof.dr.ing.Mircea Vlăduţiu

Doctorand
Ing. Cezar Morun

1998

CUPRINS

0 INTRODUCERE

Pentru câştigul, dar şi pentru menţinerea unui segment de piaţă, un anumit produs tehnic trebuie să posede atributul de calitate, constituind unul dintre cei mai importanţi factori de concurenţă. Calitatea este exprimată prin intermediul mai multor parametri, dintre care se distinge, în contextul lucrării de faţă, fiabilitatea. Din vasta problematică a fiabilităţii, cercetările cuprinse în prezenta teză sunt focalizate către subsistemul I/O, cu puternice accentuări în domeniul discurilor magnetice, pornind de la ideea că se poate construi un ansamblu fiabil din componente mai puţin fiabile, prin introducerea redundanţei. Prin analize de profunzime asupra celui mai vulnerabil nucleu *hardware* al unui sistem de calcul, discurile, care prin cădere pot genera cele mai costisitoare pierderi de date, în teză sunt propuse elemente de structură originale, care permit eficientizarea implementării toleranţei funcţionale la defectare.

Localizând aportul tezei, noile elemente redundante de structură propuse asigură garantarea corectei funcţionări la nivelul subsistemului I/O, cu particularizări importante pentru structurile de discuri.

0.1 Toleranţa la defectare ca scop al proiectării

Definind toleranţa la defectare, în strânsă legătură cu fiabilitatea, drept capacitatea unui sistem de a-şi îndeplini misiunea, în intervalul de timp stabilit, în condiţii de exploatare extreme şi eventual viciat de un număr limitat de subsisteme defecte.

Strategiile, conceptele şi mecanismele specifice toleranţei la defectarea sistemelor de calcul sunt sintetizate în Figura 0-1.

Câteva precizări sunt necesare pentru structura din Figura 0-1, unde prin mascarea defectelor se înţelege, de exemplu, aplicarea votării prin majoritate, dar şi corectarea unor erori la operaţii de citire/scriere asupra unui disc/canal. Altă soluţie pentru crearea toleranţei la defectare o reprezintă reconfigurarea care reprezintă procesul de eliminare a unităţii defecte din sistem şi restaurarea stării operaţionale. Apelând la reconfigurare, trebuie avute în vedere următoarele procese: detecţia defectelor reprezentând procesul de recunoaştere a anomaliei funcţionale apărute; localizarea defectului prin determinarea unităţii cu funcţionare eronată; limitarea propagării defectului prin acţiuni asupra unităţilor transportatoare de defect; restabilirea operaţională din starea de defectare prin reconfigurare.

<table>
<tr><td rowspan="6">Toleranţa la defectare</td><td>Specificarea Defectelor</td><td>Modelarea defectelor</td><td>Specificarea claselor de defecte tolerate</td></tr>
<tr><td colspan="3">Mijloace utilizate – redundanţa</td></tr>
<tr><td colspan="3">Acoperirea şi diagnoza defectelor</td></tr>
<tr><td colspan="3">Măsuri împotriva defectelor şi tratarea erorilor</td></tr>
<tr><td>Acoperirea Erorii</td><td>Compensarea erorilor</td><td>Eliminare logică a componentelor defecte</td></tr>
<tr><td></td><td>Mascarea defectelor</td><td>Reconfigurare</td></tr>
<tr><td></td><td>Corecţia erorilor</td><td></td></tr>
</table>

Figura 0-1

Pentru soluţionarea complexei sarcini de proiectare a unui sistem de calcul cu toleranţă la defectare, introducerea redundanţei se dovedeşte măsura cea mai eficientă, aceasta fiind, însă, opusa metodelor convenţionale de proiectare care tind spre rezultate care oferă un raport performanţă/cost cât mai bun (prin eliminarea redundanţei).

Prin redundanţă, în contextul prezentei lucrări, vom înţelege prezenţa în stare funcţională pregătită a mai multor mijloace tehnice decât sunt necesare pentru executarea unor sarcini specificate. Astfel, vom putea determina patru tipuri de redundanţă:

- redundanţa structurală legată strict de componente *hardware*, reprezentată de extensii ale sistemului cu componente *software* şi *hardware* suplimentare faţă de cele strict necesare execuţiei funcţiilor specificate;
- redundanţa funcţională constă într-o extensie a funcţiilor unui sistem;
- redundanţa informaţională se bazează pe utilizarea codurilor detectoare şi corectoare de erori;

- redundanţa temporală presupune un interval de timp suplimentar faţă de unul necesar funcţionării normale, care este necesar la un sistem redundant funcţional pentru executarea tuturor funcţiilor.

0.2 Toleranţa la defecte în subsistemul I/O

Subsistemul de I/O este partea cea mai vulnerabilă a unui sistem de calcul deoarece este, în mare măsură, în contact cu mediul înconjurător şi, în special, cu operatorii umani. Tot acest subsistem este cel cu evoluţia cea mai lentă din punct de vedere al performanţei şi al fiabilităţii.

În contextul considerat mai sus, se remarcă discurile magnetice, principalele mijloace de stocare de date, care printr-o funcţionare defectuoasă pot compromite total activitatea unui sistem de calcul.

Asigurarea protecţiei datelor faţă de situaţiile anormale, inclusiv erori umane, căderi de echipamente şi condiţii ostile a apărut ca o necesitate stringentă chiar cu primul sistem de calcul. *Backup*-ul, copierea periodică a datelor pe un suport mobil păstrat în afara sistemului, este soluţia esenţială practică, dar lentă, în ciuda abilităţii date de creşterea numărului de discuri utilizate pentru a preveni alterarea accesului la date corecte în situaţii de căderi sau dezastre. Şi mai puţine progrese s-au realizat în domeniul minimizării erorilor umane ceea ce impune utilizarea în continuare a procedurilor de *backup*, chiar şi pentru sistemele de discuri care pot supravieţui la căderi şi dezastre.

Necesitatea ca sistemul de discuri să menţină un acces continuu *on-line* a apărut în anii '70, o dată cu creşterea masivă a utilizării sistemelor de calcul *on-line*. *Backup*-ul şi restaurarea *off-line* erau considerate suficiente pentru a acoperi o cădere. Deoarece accesul la date corupte sau distruse nu are nici o valoare, proiectanţii au fost obligaţi să furnizeze soluţii de protejare a datelor şi a accesului la acestea în cazul unor căderi. Deoarece datele sunt memorate, în principal, pe discuri, prima soluţie de creştere a fiabilităţii, bazată pe redundanţă, a constat în memorarea aceloraşi informaţii pe două suporturi diferite *(mirroring)*, metoda care protejează datele şi menţine accesul la ele chiar şi în situaţia căderii unui disc din perechea utilizată.

Soluţia *mirroring* presupune costuri cel puţin duble pentru subsistemul de discuri, faţă de cel clasic, astfel că a apărut necesitatea practică de a găsi soluţii mai puţin costisitoare pentru creşterea protecţiei datelor şi asigurarea accesului continuu la acestea. Astfel, la sfârşitul anilor '80 s-au pus bazele conceptului de RAID – Redundant Array of Independent Disks [PDGG88]. RAID utilizează redundanţa sub forma unor informaţii de control (paritate). Faţă de *mirroring*, unde redundanţa era de 100%, la RAID ea poate scădea şi sub 10%.

Pentru o tratare unitară, soluţia *mirroring* a fost inclusă în familia RAID, care a fost divizată în două părţi:

- *mirrored* RAID
- *parity* RAID – RAID protejat prin paritate.

Spre deosebire de *mirrored* RAID, primele *parity* RAID–uri sufereau de o foarte scăzută performanţă datorată necesităţii generării şi scrierii parităţii în timpul operaţiei de scriere şi a regenerării *on-line* ca răspuns la o cerere către un disc căzut. Această problemă a fost rezolvată parţial prin tehnici precum *caching*, scriere asistată pe discuri, etc. şi a condus la o dezvoltare exponenţială a utilizării RAID-ului după mijlocul anilor '90.

Două decade de cercetări în domeniul protecţiei datelor şi menţinerii *on-line* a accesului în cazul unei căderi sau a unui dezastru au generat o largă paletă de soluţii de sisteme de discuri cu diverse grade de rezistenţă. Pentru evitarea confuziilor şi pentru a uşura alegerea celei mai potrivite soluţii pentru o anumită aplicaţie a fost înfiinţată o asociaţie internaţională a cercetătorilor din domeniul RAID la care s-au afiliat şi cei mai importanţi producători, asociaţie denumită RAID Advisory Board's (RAB).

În multe sisteme de calcul, subsistemul I/O reprezintă adesea o pondere importantă în cadrul performanţelor sistemului. De-a lungul ultimilor ani, viteza CPU a crescut în proporţii de 40% până la 100% pe an, pe când timpul căutării pe disc s-a îmbunătăţit cu doar 7% pe an (creşteri asemănătoare de performanţă se pot observa şi la celelalte componente ale subsistemului I/O).

Acest fapt a condus la o mare discrepanţă între viteza CPU/memorie principală şi cea a subsistemului operaţiilor I/O, şi este de aşteptat ca aceasta să crească şi mai mult în viitorul apropiat. Bazându-ne pe aceste observaţii, este de prevăzut ca viitoarele creşteri ale vitezei de lucru ale procesoarelor să determine doar o mică îmbunătăţire a performanţelor

totale ale sistemului de calcul. Evident, datorită limitărilor tehnologice, soluţiile pot fi găsite doar la nivel structural.

O idee atractivă este cea numită matrice de discuri *(disk array)*, unde subsistemul de discuri este compus din mai multe discuri, cu datele împrăştiate pe toate aceste discuri. Conceptele de „disc întreţesut" *(disk interleaving)* şi „disc secţionat" *(disk striping)* au fost pentru prima oară folosite de către Kim [Kim85], respectiv Salem şi Garcia-Molina [GaMo87]. De atunci, o mare cantitate de muncă a fost depusă asupra mai multor modele, referitor la performanţele matricelor de discuri şi a fiabilităţii acestora. Matricele de discuri au fost împărţite în şase clase denumite RAID Levels *(Redundant Arrays of Inexpensive Disks)* [PDGG88]. Printre cele şase, cele mai promiţătoare două candidate la obţinerea de performanţe ridicate într-un sistem de calcul par a fi RAID 1 (matrice de discuri oglindită - *mirrored disk array*) şi RAID 5 (matrice cu paritate rotită - *rotated parity array*), RAID 6 nefiind încă unanim acceptată drept RAID standard, iar RAID 7 reprezentând un rezultat al autorului acestei lucrări.

Primele mele cercetări în domeniul fiabilităţii transmiterii şi stocării datelor s-au focalizat, începând cu anul 1991, în subdomeniul RAID, în principal abordând problematica legată de „Algoritmii de distribuţie a datelor, parităţii şi spaţiilor de rezervă".

0.3 Structura tezei de doctorat

Prezenta teză de doctorat abordează problematica creşterii fiabilităţii şi performanţei transmiterii şi stocării datelor prin soluţii structurale îmbinate cu utilizarea unor mecanisme redundante de autocontrol şi autocorecţie, care să asigure o rezistenţă sporită a subsistemelor de I/O la căderi şi dezastre. În partea finala a lucrării, deschid un nou domeniu de cercetare prin generalizarea principiilor RAID către întreg sistemul de calcul, ceea ce, în opinia mea, va conduce în viitor la un mai mare grad de unificare a algoritmilor utilizaţi în scopul creşterii fiabilităţii şi performanţei.

Lucrarea cuprinde şapte capitole şi o listă bibliografică.

Precizez că, în vederea unei prezentări unitare, în lucrare pentru referirea elementelor, structurilor şi tehnicilor uzitate am folosit termeni şi abrevieri consacrate în limba engleză, acolo unde în limba română nu există o corespondenţă convenabilă.

După o scurtă introducere care constituie prezentul capitol, capitolul 1 cuprinde o tatonare a domeniului, precum şi o trecere în revistă a metodelor generale de creştere a fiabilităţii transmiterii şi stocării datelor.

În capitolul 2, pornind de la focalizarea iniţiala a cercetărilor mele în domeniul subsistemelor de discuri, se realizează o incursiune în domeniul metodelor consacrate de creştere a fiabilităţii acestora prin utilizarea codurilor şi a redundanţei, în special în cadrul structurilor de tip RAID. În final, prezint o clasificare a structurilor de discuri redundante din punct de vedere al fiabilităţii, în scopul clarificării problemelor existe în domeniul meu de interes.

Capitolul 3 cuprinde o analiză originală a diverselor soluţii RAID, în cadrul căreia sunt prezentate câteva dintre rezultatele originale obţinute de autor, cum ar fi: variante arhitecturale noi pentru RAID level 1 şi 5, strategii de accesare pentru RAID 1 şi 5.

În capitolul 4 propun un nou nivel RAID 7, algoritmi de distribuţie a datelor şi a spaţiilor de rezervă pentru RAID 7, etc.

În capitolul 5, autorul propune o generalizare originală a conceptului de RAID, RAIC *(Redundant Array of Independent Channels)* generalizare care deschide practic noi direcţii de cercetare, unificând abordarea din punct de vedere al performanţei şi fiabilităţii pentru toate subsistemele unui sistem de calcul. După o trecere în revistă a problemelor legate de RAIC (organizarea datelor, distribuţia datelor, mecanisme de redundanţă, tratarea căderilor etc.) se propune o clasificare a soluţiilor RAIC, analogă celei existente pentru RAID. Menţionez că numerele de niveluri nu presupun nici un fel de ierarhizare din punct de vedere al fiabilităţii sau performanţei.

În finalul capitolului 5 se prezintă o descriere a unor experimentări cu structuri RAIC şi câteva rezultate practice obţinute.

Capitolul 6 prezintă concluziile şi sinteza contribuţiilor originale aduse de autor pe parcursul lucrării.

Autorul tezei mulţumeşte în mod deosebit conducătorului ştiinţific, domnului prof.dr.ing. Mircea Vlăduţiu care, prin îndrumarea competentă într-o direcţie de un puternic interes actual, prin profesionalism în abordarea problemelor specifice şi validarea soluţiilor alese, şi nu în ultimul rând prin răbdarea şi timpul acordat de-a lungul perioadei de pregătire, a contribuit decisiv la elaborarea tezei.

Autorul mulţumeşte, de asemenea, colegilor de la Catedra de Calculatoare a Facultăţii de Automatică şi Calculatoare din cadrul Universităţii Politehnice din Timişoara, pentru sugestiile şi recomandările primite.

1 METODE DE CREŞTERE A FIABILITĂŢII ŞI DISPONIBILITĂŢII

1.1 Indicatori pentru caracterizarea fiabilităţii şi disponibilităţii

1.1.1 Clasificarea defecţiunilor

Într-un sistem există trei categorii majore de surse de erori:

- greşeli de proiectare;
- defecţiuni fizice;
- erori de interacţiune ale operatorului uman *(procedural errors)*.

Aceste surse de erori contribuie la căderile sistemului. Pe de o parte, greşelile de proiectare apar atât în *hardware*, cât şi în *software*. Acestea din urmă sunt predominante şi se elimină greu din sistem [Toy88]. Pe de altă parte, defecţiunile fizice sunt rezultatul îmbătrânirii componentelor sau a influenţei mediului, care cauzează devierea unor caracteristici ale echipamentului peste limitele câmpului specificat al toleranţei. Se produc, astfel, erori determinate de aşa-numitele defecţiuni parametrice *hardware*.

Acţiunile improprii ale operatorului la panourile de control şi de întreţinere pot determina funcţionarea defectuoasă a sistemului. Ceea ce introduce operatorul are prioritate mare, sistemele fiind foarte vulnerabile la comenzi improprii şi la erori de operare. Proiectarea unui sistem tolerant la defecţiuni necesită găsirea unei metode care să împiedice ca o componentă *hardware* deteriorată să determine o defecţiune logică şi ca aceasta să determine, la rândul ei, o eroare.

În relativ scurta istorie a designului tolerant la defecte au existat multe confuzii între definiţii şi concepte fundamentale. De exemplu, se foloseau, în mod interschimbabil, termeni ca: *defecţiune, deteriorare* şi *eroare*. Unii autori susţin că o deteriorare a avut loc atunci când calculatorul nu mai răspunde la comenzi. Dimpotrivă, alţi autori afirmă că o deteriorare este o defecţiune fizică specifică, mai degrabă, componentelor electronice. Datorită diversităţii de păreri în privinţa înţelegerii acestor noţiuni şi pentru a se evita echivocul în lectura acestei lucrări, consider utilă definirea unor termeni, care, deşi au serioase trimiteri bibliografice - [Lala85] şi [John89] -, sunt încă departe de acceptarea universală. Aceste definiri sunt:

- Deteriorarea *(failure)* apare atunci când ieşirea obţinută diferă de cea proiectată. Ieşirea poate fi privită la diferite niveluri de abstractizare: la nivel de componentă electronică, la nivel de bloc sau la nivel de sistem;
- Defecţiunea sau defectul *(fault)* este o stare eronată a maşinii sau a sistemului de programe, stare generată de: deteriorări ale componentelor, interferenţe fizice cu mediul, erori ale operatorului uman sau proiectare incorectă;
- Eroarea *(error)* este manifestarea unei defecţiuni într-un program sau într-o structură de date. Ea poate apărea la o anumită distanţă în spaţiu şi timp faţă de locul, respectiv momentul, defectării;

Definiţiile de mai sus pot fi caracterizate de următoarele adjective:

- Adjectivul „permanent“ (idem în engleză) descrie o deteriorare, o defecţiune sau o eroare cu caracter continuu şi stabil. Dacă este vorba despre echipament, atunci o deteriorare permanentă reflectă o schimbare fizică ireversibilă;
- Adjectivul „intermitent“ *(intermittent)* descrie o defecţiune sau o eroare care se repetă ocazional;
- Adjectivul „tranzitoriu“ *(transient)* descrie o defecţiune sau o eroare non-repetitivă, cauzată de condiţii temporare. Exemple sunt acţiunea particulelor α asupra memoriilor semiconductoare sau variaţia tensiunii de alimentare a echipamentelor. Astfel de erori sunt nereparabile, pentru că nu există defect fizic al echipamentului. Evitarea lor se poate face doar printr-o proiectare mai riguroasă.

Deteriorările sistemului sunt clasificate, funcţional vorbind, în defecţiuni *hardware*, erori *software* şi erori de procedură. Există relaţii strânse între cele trei tipuri de deteriorări de sistem şi categoriile surselor de defecţiuni. Defecţiunile fizice apar numai la echipament. Erorile de proiectare, chiar dacă apar în *hardware*, produc, în majoritatea lor, erori de tip *software*. Erorile de interacţiune sunt cauzate, de obicei, de greşelile făcute de operator.

În afara deteriorărilor normale ale dispozitivelor, la defecţiunile *hardware* mai contribuie condiţiile mediului înconjurător (perturbaţiile), erorile de proiectare (greşelile) şi cronografiile improprii *(timing problems)*. Greşelile de proiectare, ca în cazul erorilor software, sunt foarte greu de apreciat cantitativ. Calculele de fiabilitate se bazează mai mult pe defecţiunile dispozitivelor.

1.1.2 Estimarea fiabilităţii

Estimarea fiabilităţii este un proces de apreciere a fiabilităţii realizabile de un articol (element, subsistem sau sistem), având disponibile datele ratei de defectare, adică, prin estimarea fiabilităţii, se apreciază probabilitatea îndeplinirii obiectivelor articolului respectiv pentru o aplicaţie specifică. Aceste calcule sunt utile în stadiile de început ale unui proiect.

După îndepărtarea defecţiunilor timpurii, componentele se menţin pe o perioadă lungă la o rată de defectare aproximativ constantă. În timpul acestei perioade, rata de defectare este, de obicei, scăzută şi este puţin probabil ca defecţiunile să provină de la o singură cauză. Rata constantă de defectare, reprezentată în porţiunea de viaţă utilă, scoate în evidenţă faptul că probabilitatea de defectare este independentă de vârstă. Pentru orice rată constantă de defectare, valoarea fiabilităţii depinde numai de timp. Funcţia fiabilităţii $R(t)$ (*reliability*), care este caracterizată de o rată constantă de defectare este o distribuţie negativă exponenţială şi are forma:

$R(t) = e^{-\lambda t}$, unde λ este rata de defectare, iar t este timpul.

Timpul mediu dintre defectări (*Mean Time Between Failures*; MTBF) este timpul mediu, exprimat, de obicei, în ore, în care un articol ar putea să funcţioneze corect înainte de a se deteriora şi reprezintă măsura cantitativă a fiabilităţii [Lala85], [Toy88]. În general, MTBF-ul unui sistem ar putea fi tratat ca integrala funcţiei de fiabilitate a întregului sistem: $\int_0^\infty R(t)\,\mathrm{d}t$

În câteva cărţi de literatură, intervalul este definit ca timp mediu de defectare (*Mean Time To Failure*; MTTF). Din punct de vedere tehnic, MTTF şi MTBF nu sunt identici, MTBF-ul poate fi definit, într-o formă alternativă, astfel:

$MTBF = MTTF + MTTR$, unde MTTR este timpul mediu de reparaţie *(Mean Time To Repair)*.

Rata de reparaţie μ *(repair rate)* sau inversul timpului mediu de reparaţie, adică

$$\mu = \frac{1}{MTTR}$$

este un alt factor care afectează substanţial fiabilitatea şi mentenabilitatea unui sistem. Când o unitate dintr-un sistem duplicat este defectă, sistemul este dependent de a doua unitate pentru a-şi

continua operaţia. Dacă unitatea defectă este reparată repede, riscul deteriorării întregului sistemului este mic deoarece unitatea a doua va opera astfel încât să păstreze integritatea funcţionării sistemului. Întrucât sistemul este vulnerabil numai până la reparaţia unităţii defecte, un timp scurt de reparaţie poate creşte foarte mult fiabilitatea sistemului.

În cazul RAID, pentru estimarea fiabilităţii, nu mai sunt suficienţi MTBF, MTTF şi MTTR, parametri care descriu cu precădere caracteristicile fizice ale dispozitivelor. Astfel, ca de curând RAB [Mass97] a introdus:

- **MTDA** – *Mean Time until (Loss of) Data Availability,* timpul mediu până ce căderea unei componente cauzează o pierdere a accesibilităţii într-o matrice de discuri. Pierderea accesibilităţii nu implică pierderea datelor (pentru multe tipuri de căderi datele rămânând intacte – ex. Căderea unui controler neredundant). Evident că MTDA se poate folosi şi pentru RAIC, la fel ca şi MTDL.

- **MTDL** – *Mean Time until Data Loss,* timpul mediu până ce căderea unei componente generează o pierdere permanentă de date într-o matrice de discuri. Conceptual, MTDL este similar cu MTBF, dar ţine cont de posibilitatea ca redundanţa RAID-ului să protejeze împotriva pierderii de pentru un număr limitat de căderi de componente.

În prezenta teză de doctorat, autorul introduce trei noi indicatori necesari pentru evaluarea fiabilităţii, pentru RAID 7 şi RAIC 7:

MTDR – *Mean Time until Data is Recovered,* timpul mediu de la apariţia unei căderi de disc până când sunt regenerate toate blocurile pierdute în spaţiile de rezervă

MTRD – *Mean Time until Data is Regenerated,* timpul mediu în care RAID sau RAIC răspunde la o cerere dintr-o zonă afectată, prin regenerarea datelor utilizând redundanţa.

MTAR – *Mean Time until Array is Reconstructed,* timpul mediu de reconstrucţie (în starea iniţială) a matricei după înlocuirea discului sau canalului căzut.

1.1.3 Disponibilitatea

Disponibilitatea *A(t) (availability)* unui echipament este o funcţie de timp. Ea se poate defini ca probabilitate de funcţionare a echipamentului la momentul t, atât timp cât este folosit conform specificaţiilor.

Conceptul de disponibilitate este utilizat în măsurarea eficacităţii sistemului. Atât fiabilitatea, cât şi mentenabilitatea, îşi aduc aportul la definirea conceptului de disponibilitate. Această conexiune poate fi exprimată în felul următor [Lala85], [Toy88]:

$$A(t) = \frac{MTTF}{MTTF + MTTR}$$

unde funcţia fiabilităţii se presupune că este distribuţia exponenţială $R = e^{-\lambda t}$. Deoarece MTTF-ul este o măsură a fiabilităţii, iar MTTR-ul este o măsură a mentenabilităţii, ele se pot negocia astfel încât să se obţină o disponibilitate dată.

1.1.4 Dependabilitatea

În 1982, Laprie şi Coste au definit formal dependabilitatea ca un cadru de specificaţii de nivel înalt, care include mărimi ca fiabilitatea şi disponibilitatea, ca două faţete distincte ale specificaţiilor sistemului. Pe baza caracteristicilor unui sistem, dependabilitatea lui se descrie fie prin fiabilitate, fie prin disponibilitate. De exemplu, putem caracteriza mai bine un sistem nereparabil prin fiabilitatea lui, iar un sistem reparabil prin disponibilitatea lui.

În literatura mai recentă [John89], se menţionează că termenul de dependabilitate include, pe lângă conceptele de fiabilitate şi disponibilitate menţionate anterior, şi pe cele de siguranţă în funcţionare, mentenabilitate, performabilitate şi testabilitate. Dependabilitatea reprezintă calitatea deservirii furnizate de către un anumit sistem. Fiabilitatea, disponibilitatea, siguranţa, mentenabilitatea, performabilitatea şi, în fine, testabilitatea sunt mărimi utilizate pentru estimarea cantitativă a dependabilităţii unui sistem. Ţinta proiectării tolerante la defecţiuni este îmbunătăţirea dependabilitaţii prin înarmarea unui sistem pentru realizarea funcţiei cerute, în prezenţa unui număr dat de defecte. Totuşi, este de notat că un sistem tolerant la defecţiuni nu este neapărat de înaltă dependabilitate şi nici înalta dependabilitate nu necesită neapărat toleranţa la defectare.

1.1.5 Mentenabilitate

Sistemele de calcul, ca orice alte produse, se pot clasifica în două categorii: a) sistemele cu funcţie unică (simplă), la care prima defectare constituie şi finalul vieţii lor şi b) sistemele cu funcţie repetată sau sistemele cu reînnoire (restabilire), la care elementele defecte pot fi înlocuite cu altele bune, caz în care aceste produse au caracter reparabil. Prin mentenanţă, se înţelege ansamblul tuturor acţiunilor tehnico-organizatorice necesare, efectuate în scopul menţinerii sau restabilirii

unui produs în starea necesară îndeplinirii funcţiei cerute. Aceste acţiuni pot avea caracter corectiv (depistarea cauzei unei defecţiuni, repararea defectului prin înlocuirea elementelor defecte, verificarea corectitudinii operaţiilor de mentenanţă întreprinse) sau caracter preventiv (revizii, reglaje, verificări şi reparaţii planificate, executate în vederea evitării unor viitoare defecţiuni).

În aceste condiţii, este necesar să se exprime cantitativ aptitudinea unui produs de a fi repus în funcţiune în urma unui defect. Exprimarea se face, ca şi în cazul fiabilităţii, printr-o probabilitate în funcţie de timp. Mentenabilitatea este, aşadar, aptitudinea unui produs ca, în condiţii date, să fie menţinut sau restabilit în starea de a-şi îndeplini funcţia specificată, atunci când acţiunile de mentenanţă se efectuează în condiţii precizate şi într-un timp dat, cu procedee şi remedii prescrise. Corespunzător acestei definiţii, legătura dintre aspectul probabilistic şi cel funcţional se exprimă astfel:

$$M(t_r) = \text{Prob}(t_r \le T_r)$$

unde t_r este timpul de restabilire, T_r este o limită impusă duratei de restabilire, iar $M(t_r)$ este funcţia de mentenabilitate.

Relaţia între mentenabilitatea M şi rata de reparaţie μ sau timpul mediu de reparaţie MTTR este următoarea [Lala85]:

$$M(t_r) = 1 - e^{-(\mu \cdot t_r)} = 1 - e^{-\left(\frac{t_r}{MTTR}\right)}$$

La fel ca fiabilitatea, testabilitatea, dependabilitatea etc., mentenabilitatea trebuie planificată şi estimată sau evaluată încă din fazele cele mai timpurii ale conceperii unui produs.

1.1.6 Cerinţele fiabilităţii aplicate

Cerinţele fiabilităţii variază considerabil de la o aplicaţie la alta. De exemplu, obiectivele fiabilităţii unui oficiu telefonic cu comutare digitală, proiectat pentru aplicaţii telefonice, sunt:

- timpul total în care sistemul rămâne neoperativ să nu depăşească trei minute pe an, cu un timp mediu de reparaţie (MTTR) de patru ore;
- în timpul funcţionării să nu se piardă sau să nu se trateze incorect mai mult de 0,01% dintre apeluri.

Funcţionarea satisfăcătoare nu înseamnă, în acest caz, o fiabilitate de 100%. Sunt permise, totuşi, câteva apeluri incorecte, avându-se în vedere că utilizatorul va forma din nou numărul dorit, obţinând corecţia erorii. Pe de altă parte, o defecţiune într-un echipament critic, cum ar fi

transponderele de telecomunicaţii de pe sateliţii artificiali, ar determina căderea unor întregi sisteme, ceea ce este inadmisibil. În astfel de cazuri, funcţionarea satisfăcătoare impune o fiabilitate de 100%.

Durata timpului de funcţionare pentru un sistem de comutare telefonică este, pur şi simplu, durata de viaţă a echipamentului care a fost folosit în sistem. Sistemul de comutare trebuie să funcţioneze continuu, fără întreruperi, până când echipamentul este înlocuit la sfârşitul vieţii lui sau este îndepărtat pentru orice alt motiv. Deoarece servirea trebuie să fie asigurată 24 de ore pe zi, nu poate să existe timp programat pentru reparaţii sau pentru întreţinere, timp în care sistemul să fie neoperaţional. Obiectivul fiabilităţii de trei minute de neoperare pe an (două ore de nefuncţionare în patruzeci de ani [Prad86]) şi MTTR-ul de patru ore pot fi traduşi într-o disponibilitate de sistem de 99,9995%.

În contrast, aplicaţii critice pentru controlul avioanelor, ca şi în cazul lui SIFT *(Software-Implemented Fault Tolerance)* şi FTMP *(Fault-Tolerant MultiProcessor)* pretind fiabilitate ultra-înaltă. Parametrii proiectării cer ca sistemul să treacă prin mai puţin de 10^{-9} deteriorări în timpul unei misiuni de 10 ore cu un MTTR de 10 ore. Factorul de disponibilitate este egal, în această situaţie, cu 99,9999999%.

Cerinţele fiabilităţii determină într-un grad mai mare arhitectura sistemului, tipul şi cantitatea de redundanţă precum şi costul sistemului. Redundanţa dual se dovedeşte a fi adecvată aplicaţiilor telefonice. Pe de altă parte, pentru controlul critic al avioanelor, triplarea cu rezervă este necesară pentru a achita un grad mare de fiabilitate.

1.1.7 Tehnici de redundanţă

Defecţiunile *hardware* pot afecta secvenţele de comandă sau cuvintele de date care se află în interiorul maşinii. Acest lucru duce la două tipuri de erori:

- secvenţa programului este neschimbată, dar defecţiunea a afectat rezultatele finale;
- secvenţa programului este schimbată, iar programul nu mai execută algoritmul specificat.

Erorile *software* sunt rezultatul unor traduceri (translatări) greşite sau ale unor implementări improprii ale algoritmilor originali şi, de asemenea, deviază execuţia instrucţiunilor de la secvenţa corectă. În multe situaţii, din păcate, defecţiunile *hardware* nu pot fi distinse de cele *software*. Din acest motiv, sistemele trebuie să fie tolerante la defecţiuni, indiferent de sursa acestora.

O modalitate bună de a produce calculatoare tolerante la defecte este aceea de a introduce redundanţa prin multiplicarea părţilor lor. Redundanţa permite calculatoarelor să ocolească erorile, în felul acesta rezultatele fiind corecte. Această metodă este cunoscută ca redundanţă de protecţie, fiind compusă din redundanţe *hardware*, *software* şi de timp. Redundanţa *hardware* este constituită din componente adiţionale, care detectează şi corectează erorile. Redundanţa *software* conţine programe adiţionale, care au capacitatea de a restabili funcţionalitatea sistemului, în condiţii problematice. De asemenea, mai poate conţine programe de detecţie a erorilor, de diagnoză şi de autocontrol, prin care să testeze periodic toate circuitele logice ale calculatorului. Redundanţa temporală este obţinută prin repetarea unei operaţii eronate, imediat după repetarea unei erori. Redundanţa temporală se implementează adesea prin *hardware*. De exemplu, logica *hardware* poate iniţia recitirea automată a unei locaţii de memorie în care s-a detectat o eroare de paritate.

Chiar dacă redundanţa de protecţie este clasificată funcţional în trei tipuri diferite, în practică ele se contopesc adesea, pentru asigurarea unei fiabilităţi sporite. În cazul redundanţei *software*, programul de control are nevoie atât de spaţiu de memorie *(hardware)*, cât şi de timp de execuţie.

Fiecare dintre aceste tipuri de redundanţă, precum şi diferitele lor combinaţii, au fost folosite în proiectarea calculatoarelor tolerante la defecţiuni. Prioritatea acordată la implementare unuia dintre tipurile de redundanţă enumerate mai sus se face în funcţie de tipul aplicaţiei şi de restricţiile impuse de costuri, de cerinţele de timp şi de cerinţele de fiabilitate.

1.2 Metode de autocontrol, implementate *hardware*

Atât structura redundanţei dinamice cât şi cea a redundanţei statice, descrise mai sus, sunt metode care folosesc părţi de rezervă pentru a face sistemul capabil să tolereze defecţiunile. Atunci când se foloseşte redundanţa statică, unităţile de rezervă (circuite, componente sau subsisteme) sunt părţi permanent active ale sistemului. Ele corectează erorile sau le maschează împiedicând, astfel, propagarea lor în sistem. Funcţia mascării intervine automat, iar acţiunea de corectare este imediată şi încorporată *(wired in)*. Tipuri statice de redundanţă au fost folosite, mai întâi, în aplicaţiile militare care pretindeau o înaltă fiabilitate pentru o perioadă scurtă de timp, iar, mai recent, în aplicaţiile comerciale. Redundanţa dinamică, la care subsistemele adiţionale

servesc ca rezerve în interiorul sistemului, s-a folosit în aplicaţiile comerciale.

Componentele majore ale tolerării defecţiunilor sunt: detectarea erorii, diagnosticarea erorii (izolarea), restabilirea şi repararea. În cazul redundanţei dinamice, cea mai importantă componentă este detectarea erorii. Dacă toate erorile ar fi detectate şi s-ar aplica tehnicile corespunzătoare de restabilire, atunci nici o defecţiune n-ar conduce sistemul la deteriorare. Din păcate, această acoperire nu se poate realiza în practică.

Viteza detectării erorii influenţează procesul de localizare a defecţiunii şi stăpânirea ei. De asemenea, este important să se realizeze cât mai repede următorul pas, mânuirea sau izolarea defecţiunii, astfel încât defecţiunea să nu se propage. Detectarea întârziată poate deforma date importante în tot sistemul. Viteza detectării este, de asemenea, importantă pentru localizarea sursei de erori: cu cât întârzierea este mai mare, cu atât este mai greu să se găsească sursa. Diagnosticarea incorectă împiedică rezervele funcţionale de la înlocuirea corectă a unităţilor defecte. Mai mult decât atât, viteza de detectare afectează direct revenirea sistemului.

În general, detecţia erorii este realizată prin utilizarea de *hardware*, *firmware* şi *software*. Tipul schemelor de control utilizate depinde atât de structura logică a maşinii cât şi de utilizarea operaţională şi funcţională a datelor şi a semnalelor de comandă.

Schemele *hardware* de detectare a erorii încorporate într-un sistem de calcul pot avea mai multe forme. Majoritatea acestor tehnici se încadrează în clasificarea următoare:

- controale prin copii *(replication checks)*;
- controale de codificare *(coding checks)*;
- controale de temporizare *(timing checks)*;
- controale de excepţii *(exception checks)*.

Detectarea erorii poate fi amplasată strategic în interiorul unei unităţi funcţionale sau la interfaţă. Este mai avantajos dacă detecţia erorii se face intern, la stadiul ei timpuriu, în timpul funcţionării sistemului care generează rezultatele.

Controalele interne sau timpurii minimizează cantitatea activităţii sistemului şi tranziţiile eronate cauzate de o defecţiune. Astfel, nu este timp suficient pentru a se propaga defecţiunea în interiorul sistemului,

iar acţiunile necesare pentru izolarea ei şi pentru revenire la normal vor fi, probabil, mai simple. Pe de altă parte, controalele de interfaţă sau cele de ultimul moment sunt activate înainte de a se transmite orice fel de rezultat de la o unitate funcţională la alta. Acest fapt împiedică propagarea erorilor în exterior spre o altă unitate funcţională şi simplifică cea mai dificilă problemă, şi anume revenirea globală. Eroarea este conţinută în interiorul nivelului în care a fost detectată.

1.2.1 Controlul prin copii

Controlul prin copii este una dintre cele mai complete metode de detectare a erorilor dintr-un sistem de calcul. Din cauza echipamentului *hardware* necesar, această metodă este, de asemenea, şi cea mai costisitoare tehnică de redundanţă. Numărul copiilor dintr-un sistem nu trebuie să fie limitat la doi. Atunci când trei sau mai multe copii ale unui sistem sunt implicate, controlul comparării este referit ca votare. Votarea asigură suprimarea unei ieşiri eronate prin mascarea în favoarea majorităţii ieşirilor. Un exemplu de controlare prin copii într-un sistem de redundanţă triplă, modulară este FTMP *(Fault Tolerant Multiple Processor)* dezvoltat pentru calculatoarele avioanelor.

Controalele prin copii bazate pe copii identice ale unui sistem nu detectează consecinţele greşelilor de proiectare, deoarece sunt afectate toate copiile unităţii. Din acest motiv, este necesară apelarea la tehnici formale de verificare a proiectării astfel încât să se asigure că nu există greşeli de proiectare în structură care ar putea ocoli protecţia realizată de copiere.

În [MaMc88], Mahmood şi McCluskey prezintă câteva tehnici de detecţie a erorilor bazate pe utilizarea unui procesor *watchdog*. Acest procesor, care poate fi numit coprocesor de control, este mai puţin complex decât procesorul principal şi detectează erorile prin monitorizarea comportamentului sistemului. Informaţia furnizată către coprocesorul de control poate fi legată de comportamentul acceselor la memorie, cursul comenzilor, semnalele de comandă sau chiar de justificarea rezultatelor. Asemenea duplicării, pentru detecţia erorilor, această tehnică nu depinde de modelul de defecţiuni adoptat şi are, în schimb, avantajul că necesită mai puţin *hardware*.

1.2.2 Controale de codificare

Codurile detectoarelor de erori se formează prin adăugarea unor biţi de control la un cuvânt de date. În acest caz, capacitatea de detecţie a erorilor este direct proporţională cu numărul de biţi de control incluşi într-un cuvânt. Există două tipuri de controale de codificare: separabile

şi neseparabile. În literatura de specialitate, ele mai poartă denumirea de controale de codificare sistematice şi nesistematice. Controalele separabile *(separable checks)*, cum sunt controlul de paritate şi codurile aritmetice, sunt caracterizate prin adăugarea biţilor de control la cuvintele de date. Controalele neseparabile *(nonseparable checks)*, cum este codul m-din-n, sunt codificate în formate specializate.

1.2.3 Controale de temporizare

Controlul de temporizare este o formă eficace de a controla *software*-ul pentru detecţia erorilor în programele duplicate, atunci când specificaţia unei componente include constrângeri de timp.

Un temporizator *hardware*, numit şi *watch-dog timer*, este folosit în sisteme de comutare telefonică pentru apărarea împotriva erorilor de programare din cauza cărora sistemul nu-şi mai poate reveni. Conceptul este relativ simplu. Un ceas *hardware* merge continuu în procesor şi este resetat periodic de către programul principal dacă nu se întâmplă nimic neobişnuit care să devieze programul din secvenţa lui normală de execuţie.

Dacă dintr-un anumit motiv (eroare *software* sau defecţiune *hardware*) secvenţa execuţiei nu se mai întoarce la programul principal, ceasul nu se resetează. În această situaţie, ceasul pune în circulaţie o cerere de întrerupere de înaltă prioritate şi întreprinde acţiunile necesare reiniţializării sistemului. În cazul în care există în sistem încă un procesor *stand-by*, întreruperea ceasului poate produce comutarea automată a controlului pe maşină *stand-by*, într-o tentativă de revenire din eroarea de *time-out*.

Controalele de temporizare asupra funcţionării unui circuit nu sunt controale complete pentru sistem. Ele relevă prezenţa defecţiunilor (erorilor), dar nu şi absenţa lor. Ca urmare, arhitecţii calculatoarelor folosesc controalele de temporizare ca să suplimenteze alte controale şi ca să acopere un procent mai mare de defecţiuni ale unui sistem.

1.2.4 Controale de excepţie

Programele rulează în medii protejate folosind seturi de constrângeri prestabilite. Dacă programele nu conţin erori, ele ţin cont de aceste constrângeri şi îşi realizează în mod corespunzător funcţiile specificate. Totuşi, greşelile de proiectare din *software* violează, adesea, aceste constrângeri, influenţând, în mod nefavorabil, întregul sistem. Unele circuite *hardware* de detecţie sunt adesea proiectate în sistem pentru a recunoaşte erorile de proiectare şi pentru a le trata ca excepţii. Deci,

tratarea excepţiilor se referă la detectarea şi „mânuirea“ evenimentelor anormale şi nedorite.

Constrângerile pot fi atribuite structurii *hardware* sau celei *software*. Exemple de constrângeri *hardware* sunt: alinierea improprie de adrese, locaţii de memorii neechipate, opcod neutilizat, depăşirea stivei etc. Structura *software* pune, de asemenea, câteva constrângeri sistemului pentru a-i îmbunătăţi rezistenţa şi pentru a furniza un mediu protejat programelor de aplicaţii. Câteva controale *software* de excepţie sunt: execuţia ilegală a instrucţiunilor privilegiate, depăşirea de adresă *(out of range)*, violarea memoriei *(memory acces violation)*, operanzi ilegali, operaţii aritmetice necorespunzătoare, depăşirea sau împărţirea cu zero. Detectarea unei erori iniţiază o excepţie care este urmată de invocarea automată a subrutinei de mânuire a excepţiei respective. În multe cazuri, excepţia este atribuită unei greşeli de proiectare a programului.

1.2.5 Restabilirea erorii

Restabilirea este cea mai complexă şi mai dificilă funcţie pentru toate sistemele din cauza multitudinii stărilor posibile ale sistemului care pot să apară sub condiţii problematice [Toy88]. Neajunsurile proiectării *hardware* sau *software* de a detecta erorile atunci când apar au un efect direct asupra abilităţii sistemului de a se restabili. Când erorile continuă să fie nedetectate, sistemul rămâne defect până când problema este recunoscută. Un alt tip de restabilire apare atunci când sistemul nu este capabil să izoleze corespunzător un subsistem defect şi să reconfigureze, în jurul lui, un sistem operaţional. Detectarea cât mai rapidă a unei erori uşurează determinarea componentei defecte şi restabilirea sistemului. Deci, detectarea rapidă a erorii şi izolarea ei sunt condiţiile fundamentale ale restabilirii.

1.2.6 Clasificarea procedurilor de restabilire

Există trei clase de proceduri de restabilire: restabilirea totală *(full recovery)*, restabilirea degradată *(degraded recovery)* şi închiderea pentru salvare *(safe shutdown)*. Procedurile de restabilire pot fi invocate automat (fără nici o interacţiune între operatorul de întreţinere şi sistem) sau manual. Restabilirile iniţiate manual fac uz de secvenţe vaste, controlate prin program.

1.2.6.1 Restabilirea totală

Într-o facilitate de timp real, sistemul care furnizează servicii în mod continuu trebuie să rămână operaţional chiar şi într-un mediu defect. Aceasta înseamnă că simptomele trebuie să fie recunoscute repede şi

unităţile defecte trebuie să fie reparate cu puţină sau chiar fără intervenţie din partea utilizatorului.

Pentru a dispune de sistem, tot timpul şi la întreaga sa capacitate, subsistemele corespunzătoare de schimb trebuie să fie disponibile ca unităţi de înlocuire pentru cele defecte.

O procedură de restabilire totală are nevoie, în mod obişnuit, de toate cele cinci aspecte ale calculării tolerante la defecte: detectarea erorii, izolarea erorii, restabilirea sistemului, diagnosticarea erorii şi repararea ei. Perioada de întrerupere a servirii, în timpul procesului de restabilire, este minimizată şi, de obicei, nu este observată de către utilizator.

Diagnosticarea şi repararea, care consumă cel mai mult timp în cadrul procedurii de restabilire, pot fi amânate şi intercalate printre operaţiunile normale ale sistemului.

Restabilirea degradată

Ca şi în cazul restabilirii totale, toţi paşii implicaţi în tolerarea defecţiunii (detectare, izolare, restabilirea sistemului, diagnoză şi reparare) trebuie să fie incluşi în procedură. Aceeaşi secvenţă de evenimente apare la procedura degradată, exceptând faptul că, în acest caz, nu se cuplează alt subsistem în locul celui defect. În schimb, componenta defectă este scoasă din funcţie şi sistemul se întoarce la o stare operaţională fără defecţiuni. Funcţiile de calculare selectate sunt lăsate să opereze în sisteme care utilizează proceduri de restabilire degradată, iar din această cauză caracteristicile performanţei lor în timp real scad sub un standard normal acceptat până când se realizează reparaţiile corespunzătoare.

Închiderea pentru salvare

Aceasta este, adesea, denumită şi operaţie de salvare din malfuncţionare *(fail-safe operation)*. Ea apare ca un caz limită al restabilirii degradate, când capacitatea de calcul a sistemului a scăzut sub limita minimă acceptabilă a operaţiei. Scopurile închiderii pentru salvare sunt:

- evitarea distrugerii elementelor sistemului sau a modulelor *software* stocate (programe sau date) care mai pot fi compromise după ce a intervenit operaţia de restabilire degradată;
- încetarea interacţiunii cu orice sistem asociat (utilizator) într-un mod ordonat;
- distribuirea diagnosticului şi a mesajelor de închidere la utilizatori, la anumite sisteme şi la personalul de întreţinere.

Categoria procedurii de închidere pentru salvare este similară unui sistem fără redundanţă. Acţiunea de izolare a defecţiunii trebuie să fie iniţiată pentru a se determina identitatea ei şi locul în care se află. Funcţionarea normală a sistemului, care a fost întreruptă la momentul detectării defecţiunii, trebuie să fie, acum, suspendată datorită diagnozei şi reparării. Sistemul trebuie să fie restabilit, atât din punct de vedere *hardware*, cât şi *software*, astfel încât să-şi poată relua procesarea normală.

1.2.6.2 Reconfigurarea

În sistemele redundante, un ansamblu de piese de schimb sau de unităţi multifuncţionale asigură continuitatea funcţionării sistemului. Structura cea mai simplă este configuraţia duplex, în care fiecare unitate funcţională este duplicată. Dacă una dintre unităţi se defectează, unitatea duplicată se cuplează păstrând, astfel, continuitatea funcţionării, în timp ce unitatea defectă se repară. În timpul perioadei de reparaţie poate să apară o defecţiune în unitatea duplicată, caz în care întregul sistem se va deteriora. Dacă intervalul de reparaţie este scurt, atunci probabilitatea de defectare, simultană, a două unităţi identice este foarte mică.

Configuraţia duplex cu o singură unitate poate fi partiţionată într-o configuraţie duplex multi-unitate. În acest aranjament, fiecare subunitate conţine un număr mic de componente care pot fi comutate într-un sistem în lucru. Sistemul va pica numai dacă o eroare apare în subunitatea redundantă, în timp ce subunitatea originală este supusă reparaţiei. Din moment ce fiecare subunitate conţine câteva componente, probabilitatea de apariţie a defecţiunilor simultane într-o pereche dublă de subunităţi este redusă.

1.3 Metode de autocontrol implementate *software*

Necesitatea dezvoltării mecanismelor *software* pentru detectarea erorilor de programare este foarte acută atunci când se pune problema construirii unor sisteme fiabile. Mecanismele *hardware*, chiar dacă sunt foarte avansate din punct de vedere tehnologic, nu sunt specializate în analiza erorilor *software*.

Spre deosebire de ratele de deteriorare ale componentelor *hardware*, despre comportamentul şi distribuţia erorilor de programare nu există statistici de încredere. Detecţia erorilor *software* se poate realiza numai prin recunoaşterea comportamentului anormal al sistemului.

Pentru detectarea eventualelor devieri, este nevoie de un set de standarde pentru descrierea comportamentului normal. Prin controlarea

motivaţiilor comportării programului în anumite stadii ale calculării, se reuşeşte detectarea rapidă a erorilor şi limitarea propagării datelor eronate.

Există mai multe tehnici generale pentru urmărirea comportamentului unui sistem de calcul şi detectarea devierilor de la comportamentul normal. Printre acestea sunt: funcţia unui proces, secvenţa de comandă a unui proces, date de proces. Un proces este definit aici ca fiind o porţiune de calcule de sine stătătoare, care, o dată iniţiată, se realizează complet fără a mai avea nevoie de intrări adiţionale.

1.3.1 Funcţia unui proces

Motivarea ieşirilor pentru un set dat de intrări este controlată la nivelul blocului funcţional. Anumite măsuri de performanţă pot fi folosite pentru a se indica funcţionarea corespunzătoare a sistemului.

Atunci când încărcarea aplicată este normală, dar măsurătorile care caracterizează performanţă sistemului (timpul de răspuns, debitul şi timpul necesar pentru a se realiza o funcţie standard) sunt în afara valorilor limită, sistemul are, probabil, una sau mai multe erori.

1.3.2 Secvenţa de comandă a unui proces

Secvenţa de calcule făcută de către un proces care se află în execuţie este denumită secvenţă de comandă *(control sequence)*. Fiecare calcul schimbă multe stări ale sistemului şi detectează multe dintre neregulile acestuia.

Este evident că *hardware*-ul necesar calculului secvenţei de comandă trebuie să fie cât mai fiabil, pentru a nu constitui el însuşi o sursă suplimentară de erori sau de defectări.

1.3.3 Date de proces

Integritatea datelor unui sistem şi structura lui pot fi observate în timp ce sistemul execută secvenţa programului. Deteriorarea datelor poate fi cauzată atât de o defecţiune *hardware*, cât şi de o eroare *software*.

1.3.4 Restabilirea *software*

Obiectivul restabilirii după o eroare este revenirea sistemului la o stare consecventă şi la o funcţionare corespunzătoare.

Există trei strategii de restabilire:

- restabilirea cu întoarcere *(backward error recovery)*;
- restabilirea cu salt înainte *(forward error recovery)*;

- reinițializarea *(reset*-ul).

1.3.5 Diagnosticarea erorii

Ținând cont că detectarea erorii determină corectitudinea funcționării unui circuit, diagnoza erorii localizează defecțiunea la o unitate înlocuibilă. Unitatea înlocuibilă poate fi o componentă, un circuit sau un subsistem. Rutina de diagnosticare a erorii utilizează *hardware* pentru detectarea erorii şi secvențe de test în scopul localizării unității defecte. Dacă sursa erorii o constituie o singură entitate, atunci defecțiunea este corectată simplu, prin înlocuirea entității respective, fără să mai fie necesară diagnoza. Dacă circuitul de detecție este mai puțin specializat, atunci rutina de diagnosticare a erorii poate fi solicitată la izolarea, în continuare, a unității afectate.

Cele mai dificile sarcini ale proiectării întreținerii sunt: restabilirea sistemului şi diagnosticarea. Eficacitatea lor poate fi determinată prin simularea modului de comportare a sistemului în prezența unei anumite defecțiuni. Prin intermediul simulării deficiențele proiectării pot fi identificate şi corectate înainte de a se da sistemul spre utilizare. Este necesar să se evalueze abilitatea sistemului la detectarea erorilor, la revenirea automată la funcționarea normală şi la furnizarea informației de diagnoză (de exemplu: locația defecțiunii). Simularea, fizică sau digitală, a defecțiunii este un aspect important al proiectării întreținerii.

1.3.6 Autocontrolul ca mijloc de creştere a fiabilităţii şi disponibilităţii sistemelor de calcul

1.3.6.1 Metode de autocontrol aplicate la diferite niveluri de structură

1.3.6.1.1 Caracteristicile metodelor de proiectare structurată la nivel de bistabil

Progresele recente din tehnologia VLSI au avut o influență mare asupra testării sistemelor digitale (numerice). Sistemele digitale de astăzi au de la o sută de mii până la un milion de porți de logică aleatoare şi celule de memorie ceea ce face ca generarea testului şi simularea defecțiunilor să fie extrem de dificile. Chiar dacă se pot folosi maşini *hardware* dedicate sau super-computere pentru a genera teste eficace, costul acestora face imposibilă aplicarea lor în practică. Pentru a se depăşi această problemă, cercetarea s-a îndreptat către găsirea altor metode de testare a sistemelor digitale, incluzând proiectarea pentru testabilitate şi generarea de test la nivel funcțional (simularea defecțiunilor).

Dintre toate tehnicile propuse până în prezent, proiectarea pentru testabilitate (DFT, *Design For Testability*) este cea mai renumită, iar dintre toate tehnicile DFT, cea mai folosită este tehnica scanării *(scan-design)*. În tehnica scanării (în [Marc93] i se spune tehnică sau metodă SCAN), bistabilele sunt conectate într-un circuit serial şi sunt folosite ca terminale de tip I/O. Prin aplicarea tehnicii de scanare putem transforma un circuit secvenţial în unul combinaţional, făcând astfel mai simplă generarea *pattern*-urilor de test pentru circuit. De asemenea, se poate partiţiona logic circuitul în câteva subcircuite şi se pot genera, pentru fiecare în parte, *pattern*-urile de test.

Firma NEC a aplicat cu succes tehnica proiectării de scanare la sistemele comerciale de calcul încă din anul 1968. Tehnica numită cale de scanare *(scan-path)* a fost implementată de la nivel de capsulă la nivel de sistem şi a devenit o ustensilă puternică pentru testarea şi diagnoza sistemelor de calcul VLSI. Conceptul *scan in/scan out*, care stă la baza tehnicii de scanare, a fost introdus pentru prima dată în anul 1964 de către Carter s.a. pentru dezvoltarea testelor de localizare a defecţiunilor la sisteme IBM 360. NEC a dezvoltat metoda căii de scanare pentru a rezolva problemele din anii 70, cum ar fi utilizarea intensivă a MSI/LSI din sistemele comerciale de calcul, precum şi numărul mare de porţi aflate pe plăcile acestor sisteme.

Metoda căii de scanare pare să fie o soluţie promiţătoare pentru rezolvarea acestor probleme deoarece permite mai multă controlabilitate şi observabilitate asupra circuitului supus testării. În plus, se pot testa atât circuitele combinaţionale cât şi cele secvenţiale. Conceptele de controlabilitate şi observabilitate se definesc în felul următor [Lala85], [Prad86], [John89]:

- **Controlabilitatea** arată cât de uşor se poate produce un semnal arbitrar, valid la intrările unei componente (subcircuit) prin excitarea intrărilor primare ale circuitului. De fapt, ea reprezintă abilitatea de a comanda un semnal doar prin intermediul intrărilor primare. O linie care poate fi poziţionată pe „1" logic se numeşte 1-controlabilă, altă linie care poate fi poziţionată pe „0" logic se numeşte 0-controlabilă, iar linia care poate avea ambele stări logice se numeşte complet controlabilă;
- **Observabilitatea** arată cât de uşor se poate determina la ieşirile primare ale circuitului ceea ce se întâmplă la ieşirile unei componente (subcircuit). Cu alte cuvinte, ea este abilitatea activării unei căi de la semnalul cercetat până la un punct măsurabil.

Prin componente se înţeleg fie circuite integrate standard (SSI şi MSI) pentru circuite la nivel de placă, fie celule standard de module de bibliotecă pentru circuite LSI şi VLSI. De asemenea, se presupune că mai multe componente sunt conectate cu legături unidirecţionale.

Circuitul testat prin metoda căii de scanare poate fi partiţionat logic în mai multe subcircuite, iar programele de test pot fi generate, independent pentru fiecare subcircuit. Costul total al întregului circuit scade la α^2 / n, unde n este numărul de subcircuite, iar α este rata de suprapunere . Tabelul următor arată o comparaţie între mărime şi cost.

Denumire	Mărime	Cost
Circuit original	1	1
Subcircuit	α / n	$(\alpha / n)^2$
Circuit expandat	α	α^2 / n

Figura 1-1

Evident că α depinde de n şi, din acest motiv, proiectanţii trebuie să decidă cu atenţie numărul de subcircuite necesare. Pentru un circuit bine proiectat, care este pregătit pentru partiţionare, valoarea lui α este sub 1,7. Calea de scanare poate facilita localizarea defecţiunii pentru că procesorul de diagnoză poate să urmărească uşor starea internă a sistemului. *Hardware*-ul adiţional este foarte puţin (de la 4% la 10%) şi este potrivit pentru un calculator VLSI deoarece necesită relativ puţini pini adiţionali de I/O.

Bistabilul transformat cu cale de scanare, utilizat ca şi circuitul de bază, include două tipuri de semnale de *clock*, C_1 pentru operaţia normală şi C_2 pentru operaţia de deplasare. Astfel, bistabilele unei plăci logice sunt conectate în serie printr-o cale de scanare şi operează ca un registru serial de deplasare utilizând *Clock* II, intrarea de test *(scan in)* şi ieşirea de test *(scan out)*. Utilizând o adresă de selecţie a plăcii logice, putem selecta o anumită placă dintr-o unitate logică.

Chiar dacă tehnica scanării este o tehnică DFT puternică, pentru testarea logicii aleatoare, aplicarea ei poate pune probleme în diferite tipuri de circuite. Un astfel de tip sunt şirurile de memorie încorporate în circuite de logică aleatoare. Aplicarea directă a tehnicii de scanare la fiecare

celulă de memorie din şir costă foarte mult. O soluţie posibilă este aplicarea tehnicii de scanare la un anumit cuvânt din şirul de memorie şi accesarea cuvântului fixat în timpul testării circuitului. Oricum, din moment ce, în timpul testării, memoria funcţionează ca un registru, ea trebuie testată utilizând altă procedură.

Tehnica scanării s-a dovedit foarte eficace pentru proiectarea circuitelor bipolare, dar este greu de aplicat la un circuit MOS şi mai ales la un circuit MOS dinamic. Dificultatea apare, în primul rând, din cauza *hardware*-ului adiţional. Un bistabil MOS dinamic este limitat la câteva tranzistoare şi supraîncărcarea *hardware*-ului adiţional este mare pentru un circuit VLSI MOS.

În 1977, IBM introduce tehnica LSSD *(Level Sensitive Scan Design)* care include tehnica scanării şi, în plus, impune ca toate schimbările de stare să fie controlate de nivelul semnalului de ceas şi nu de front *(Level Sensitive Design)* [Russ85], [Lala85], [Yarm90]. Această abordare reduce dependenţa funcţionării de timpii de propagare, eliminând cursele sau hazardul. Celula de bază este elementul de memorare a informaţiei, dependent de nivel, care asigură şi tratarea semnalului de scan în modul test, element numit SRL *(Shift Register Latch)*.

1.3.6.1.2 Caracteristicile metodelor de proiectare structurată la nivel de registru

Deşi s-au scris foarte multe despre standardul de test *boundary-scan* (IEEE 1149.1 *Standard Test Acces Port and Boundary-Scan Architecture*), doar câtorva dispozitive li s-a implementat această arhitectură. Acordarea completă cu acest standard, ceea ce înseamnă că proiectanţii ar putea să folosească orice dispozitiv, cu asigurarea că este compatibil cu *boundary-scan*, va apare numai după câţiva ani. Până atunci, plăcile cu cablaj imprimat vor fi hibride, conţinând circuite (dispozitive) scanabile (după standardul IEEE) şi dispozitive nescanabile. Aceste plăci necesită metodologii de testare bazate pe ambele tehnici: pat de cuie *(bed of nails)* şi *boundary-scan*.

Testarea cu *boundary-scan* poate micşora drastic timpul de dezvoltare al sistemelor complexe. De asemenea, ea oferă posibilitatea testării sistemelor care utilizează componente cu montare la suprafaţă de mare densitate şi plăci multistrat complexe. Astăzi, aceste sisteme se pot testa cu ajutorul tehnicii patului de cuie, dar există probleme din cauza geometriei micşorate a pinilor, geometrie impusă de tehnologia montării la suprafaţă. Şi în acest caz, având unelte potrivite şi dispozitive de

susţinere pentru *boundary-scan*, toată placa poate fi testată complet, folosind numai calea *boundary-scan*.

Pe de altă parte, patul de cuie şi, în general, metodele de testare în circuit *(in-circuit)* oferă câteva avantaje importante spre deosebire de *boundary-scan*. Un astfel de avantaj cheie al testării în circuit este abilitatea diagnozei deteriorărilor multiple la o singură trecere prin test. În plus, detectarea scurt-circuitelor poate fi realizată înainte de punerea sub tensiune a plăcii, reducând posibilitatea deteriorării catastrofale cauzate de acestea între pini şi tensiunea de alimentare. De asemenea, testele în circuit permit realizarea simultană a testării parametrice cu cea digitală.

Boundary-scan, în general, nu se referă la testarea dispozitivelor analogice sau pasive. Contrar impresiei larg răspândite, *boundary-scan* este compatibil cu testarea în circuit şi folosind o combinaţie a celor două se poate simplifica semnificativ testarea plăcilor.

Dispozitivul XC4005 FPGA este primul dintr-o familie de circuite compatibile cu standardul *boundary-scan*. Fiecare pin de I/O al acestui dispozitiv poate fi complet controlat şi observat prin utilizarea unor *pattern*-uri de date introduse serial în registrul de *boundary-scan* al lui XC4005 prin pinul TDI *(Test Data Input)* şi prin controlul prin pinii TMS *(Test Mode Select)* şi TCK *(Test Clock)* proveniţi din canalele ATE *(Automatic Test Equipment)*. Acest lucru este echivalent cu stimularea complexă a intrărilor pentru a obţine aceeaşi acoperire de test fără *boundary-scan*.

1.3.6.1.3 Proiectarea structurată la nivel de bloc

Încorporarea facilităţilor de test (BIT, *Built-In Test*) poate micşora considerabil timpul de întreţinere şi costurile pentru sistemele electronice complexe prin accelerarea detecţiei şi izolării defecţiunilor. La nivel de placă, testul încorporat (BIT) se realizează, de obicei, prin aplicarea unui set de *pattern*-uri pseudo-aleatoare pentru a se verifica existenţa defecţiunilor şi prin comprimarea informaţiei răspunsului de test pentru a se obţine o semnătură. Aceste tehnici sunt eficace pentru plăcile utilizate în scopuri generale atât timp cât seturile de cipuri care se testează reacţionează la cele două tehnici, dând rar semnături identice pentru plăci bune sau defecte.

Notă: În [D&TR89], McCluskey precizează că nu este recomandată utilizarea expresiei *test-response compression* ca un sinonim al expresiei *test-response compaction*, deoarece prin compactare se pierde din informaţie, în timp ce prin comprimare se elimină doar redundanţa, fără

pierderea informaţiei. În majoritatea literaturii de specialitate pe care am consultat-o - [Lala85], [John89], [Yarm90], [Fuji90a] - nu este respectată această opinie. Din această cauză, precum şi datorită faptului că în limba română, când este vorba despre reducerea volumului secvenţelor binare răspuns, este consacrat termenul de comprimare [Vlad82], [Vlad89], în continuare vom folosi denumirea „comprimare".

1.3.6.1.3.1 Autocontrolul bazat pe principiul BILBO

Tehnica de determinare a stării unui chip sau a unei plăci prin folosirea informaţiei comprimate a răspunsului de test se numeşte analiza semnăturii. În general, metodele de analiză a semnăturii pentru testarea plăcilor utilizează circuite cu observatori logici, de bloc, încorporaţi (BILBO, *Built-In-Logic Block Observers*) pentru a genera *pattern*-uri pseudo-aleatoare şi pentru a comprima răspunsurile. Deci, BILBO este o schemă încorporată de generare a testului care utilizează, în conjuncţie *scan*, design-ul cu analiza de semnături. Componenta de bază a acestei tehnici de autotest este un registru de deplasare numit registru BILBO care are patru moduri de funcţionare.

Această tehnică elimină nevoia generării şi aplicării *pattern*-urilor de test din exterior. Totuşi, este nevoie de simularea defecţiunilor pentru a se determina acoperirea de defecţiuni, realizată de secvenţe de vectori pseudo-aleatorii. Mai mult decât atât, circuitul trebuie să fie simulat pentru a se obţine valorile semnăturilor în cazul funcţionării corecte. Cantitatea datelor de test care trebuie stocate şi analizate este redusă din moment ce răspunsul circuitului la N *pattern*-uri de test este comprimat la un singur cuvânt.

1.3.6.1.3.2 Modulul comutator de testare, încorporat, pentru izolarea defectelor

În unele aplicaţii, în timpul operaţiei normale, circuitele BILBO supraîncarcă în mod nejustificat, schema din punct de vedere al consumului de energie, iar izolarea defecţiunilor durează, de obicei, mult. În locul registrului BILBO, s-a propus [KaVa89] o altă variantă de analiză a semnăturii, care utilizează un modul numit comutator de testare SW *(testing switch)*. Modulele SW pot îmbunătăţi, la nivel de placă, timpul de testare şi supraîncărcarea de putere a schemelor de analiză a semnăturii implementate utilizând circuitele BILBO.

În multe probleme de izolare a defecţiunii este mai convenabilă izolarea defecţiunii la un grup de cipuri decât la un singur circuit. Aceste grupuri se numesc grupuri de ambiguitate (AG). Modulul SW generează

pattern-uri de test pseudo-aleatorii pentru a fi aplicate la intrările unui set AG, în timp ce, simultan, comprimă datele răspunsului de test de la ieşirile unui alt set AG. Această caracteristică poate minimiza total al testului de izolare a defecţiunii.

Utilizarea modulelor SW pentru izolarea defecţiunilor introduce întârzieri adiţionale între AG-uri, în timpul modului normal de operare. Totuşi, dacă proiectanţii implementează cipuri cu comutatorul de testare utilizând tehnici de împachetare şi tehnologii avansate, pot minimiza întârzierile adiţionale. Se pare că, în cazul multor aplicaţii, această întârziere va avea un efect neglijabil asupra performanţelor sistemului.

1.3.6.1.4 Controlul erorilor la nivel de procesor

În anul 1972, Pradhan şi Reddy au arătat că, utilizând redundanţa de ordinul duplicării, codul de paritate poate realiza cu eficacitate controlarea erorilor. Afirmaţia că numai duplicarea poate realiza corecţia erorilor în procesare a fost pusă în practică, parţial, în calculatorul (4,2). Conceptul este deja utilizat ca un produs comercial prin sistemul de comutare Philips S2500 [Krol86].

S2500 utilizează patru perechi de procesoare-memorie, fiecare procesor fiind pe 16 biţi, iar memoria pe 8 biţi. De aceea, există redundanţă de ordinul patru pentru procesoare şi de ordinul doi pentru memorie. Ieşirea pe 16 biţi a fiecărui procesor este codificată într-o versiune (4,2) $GF(2^x)$ de cod R-S, producând un cod de 32 de biţi. Acest cod poate corecta orice bloc singular de 8 biţi. Fiecare codificator al ieşirii procesorului produce doar 8 biţi din cei 32 de biţi ai cuvântului de cod. Memoria asociată a celui de-al i-lea procesor stochează al i-lea octet. De aceea, codificatorul pentru procesorul i produce 8 biţi care corespund celui de-al i-lea octet. La operaţiunea de citire, toţi cei patru octeţi sunt aduşi din memorie şi decodificaţi de către decodificator, care poate corecta orice octet singular eronat sau orice eroare de doi biţi în doi octeţi diferiţi. Este de notat faptul că deteriorarea unui singur procesor sau a unei singure memorii afectează doar un singur octet din cuvântul de cod. Din acest motiv, sistemul poate tolera deteriorarea oricărei perechi singulare procesor-memorie. În plus, codul poate corecta ştergerea unui octet singular precum şi erorile singulare de bit. Astfel, dacă se demontează o pereche procesor-memorie pentru reparaţie, sistemul continuă să opereze utilizând decodificarea de erori-ştergeri *(error-erasure decoding)*. La nivel de bit poate fi corectată orice subsecvenţă de eroare singulară.

Această utilizare a codificării pentru controlarea erorii de procesor este atractivă. Singurul dezavantaj real este acela că, datorită necodificării liniilor de adrese la memorie, nu are loc corecţia de eroare la operaţia de scriere. În general, prin utilizarea căilor netradiţionale descrise mai sus, codul controlului de paritate poate furniza un control eficace al erorilor procesorului. Utilizarea lor va elimina necesitatea de conversie a codului şi întârzierea asociată în transfer procesor-memorie, furnizând astfel controlul uniform al erorii [Krol86], [Fuji90b].

1.3.6.2 Problematica construcţiei *checker*-elor de erori

Multe sisteme digitale includ detectoare de erori, adică circuite care detectează şi semnalizează apariţia erorilor. Există două motive principale pentru includerea detectoarelor de erori într-un sistem şi aceste motive sunt: prevenirea ajungerii erorii până la ieşirea sistemului şi localizarea poziţiei erorii. O cale pentru a preveni propagarea erorii este verificarea datelor recepţionate după transferul de date şi, dacă se detectează o eroare, atunci fie se cere retransmisia, fie se corectează datele utilizând un cod corector de erori (ca în memoriile RAM). O altă cale este verificarea rezultatului unei operaţii aritmetice şi repetarea operaţiei dacă rezultatul conţine o eroare detectată. Sistemele mari includ, de obicei, detectoare de erori împreună cu circuite pentru înregistrarea locaţiei şi frecvenţei evenimentelor de eroare [Mccl90]. Această informaţie este utilizată pentru a facilita întreţinerea rapidă şi exactă.

1.3.6.2.1 Circuite de verificare dublă cale

Un circuit de verificare dublă cale *(two-rail checker)*, adică un circuit care controlează dacă fiecare pereche de intrări are valori complementare, este utilizat pentru a converti cele n perechi de semnale într-o singură pereche de semnale complementare dacă şi numai dacă toate cele n perechi de intrare au semnale complementare.

1.3.6.2.1.1 *Checker* de egalitate

Un tip foarte important de *checker* este cel care verifică egalitatea, adică acela care compară două cuvinte de intrare pentru a determina dacă biţii corespunzători din ambele cuvinte au aceeaşi valoare. *Checker*-ul de egalitate este componenta cheie în *checker*-ele pentru coduri separabile şi pentru compararea ieşirilor circuitelor duplicate.

Un *checker* de egalitate cu autotestare poate fi obţinut prin complementarea tuturor biţilor din unul dintre cuvintele care trebuie să fie comparate pentru a forma un cod dublă cale *(two-rail code)*.

Checker-ele dublă cale discutate mai sus pot fi utilizate pentru a realiza un *checker* testabil de egalitate. O altă posibilitate este aceea de a concatena cuvântul complementat şi cel necomplementat pentru a forma un cuvânt de cod care are jumătate din biţii lui egali cu 1. Un checker k din $2k$ poate fi utilizat pentru a forma un *checker* testabil de egalitate.

1.3.6.2.1.2 *Checker* M din N

Checker-ele M din N formează o altă clasă de *checker*-e încorporate. Deşi sunt mai puţin importante decât celelalte, totuşi sunt utilizate. *Checker*-ul 1 din n este folosit pentru a monitoriza operarea corectă a reţelelor complete de decodificare. O implementare cu autotestare a acestui *checker* este formată prin conectarea tuturor ieşirilor decodificate care corespund intrărilor cu paritate pară la o poartă SAU şi tuturor ieşirilor care corespund intrărilor cu paritate impară la o altă poartă SAU. Ieşirile celor două porţi au fie 10, fie 01, atunci când decodificatorul operează corect. Schema este cu autotestare deoarece porţile SAU recepţionează toate configuraţiile de intrare (00, 10, 01) necesare testării lor pentru defecţiuni singulare de tip punere pe *(stuck-at)* în timpul operării normale. Această schemă poate fi convertită într-un circuit testabil cu o singură ieşire.

Codurile cu ponderea fixă *m-din-n* se utilizează atunci când se doreşte atât detectarea tuturor defectelor singulare de tip *stuck-at*, cât şi a celor unidirecţionale.

Codul zecimal 2 din 5 este cel mai cunoscut exemplu al unui cod M din N. Codurile k din $2k$ sau k din $2k+1$ sunt implementările uzuale deoarece conţin numărul maxim de cuvinte de cod pentru o lungime cunoscută a cuvântului.

1.3.6.2.2 Circuite cu autoverificare totală, încorporate

În 1968, Carter şi Schneider au ridicat obiectivele autoverificării *(self-checking)* şi au dezvoltat circuitul TSC *(Totally Self-Checking)*, care are mai multe intrări şi ieşiri. În timpul operării normale (fără defecţiuni) un circuit TSC nu recepţionează toate combinaţiile posibile la intrări, ci numai un subset al tuturor combinaţiilor posibile numit spaţiul codului de intrare *(input-code space)*. De asemenea, el nu produce toate combinaţiile posibile de ieşire, ci numai un subset numit spaţiul codului de ieşire *(output-code space)*.

Checker-ul TSC este un circuit combinaţional care simplifică sarcina unui observator. Dacă unitatea funcţională TSC produce o ieşire care aparţine spaţiului codului de ieşire, *checker*-ul TSC indică faptul că nu

există nici o eroare. O ieşire care nu aparţine spaţiului codului de ieşire se numeşte ieşire ilegală *(noncode)* şi, dacă *checker*-ul găseşte o astfel de ieşire la unitatea funcţională, atunci indică existenţa unei erori.

O situaţie interesantă apare atunci când *checker*-ul are o defecţiune. Din moment ce se presupune că unitatea are numai o singură defecţiune la un moment dat, se poate considera că, dacă *checker*-ul are o defecţiune, unitatea funcţională nu are nici o defecţiune şi, deci, nu va produce nici o ieşire ilegală. Astfel, este nevoie de un mecanism prin care fiecare defect din interiorul *checker*-ului să se releve ca o eroare la ieşirea lui. Este necesar, de asemenea, să se codifice ieşirile *checker*-ului printr-un cod detector de erori unidirecţionale. *Checker*-ul trebuie să producă un număr cât mai mic de ieşiri deoarece acestea vor fi monitorizate din exterior. Cel mai mic cod nesistematic cunoscut, numit şi cod dublă cale *(dual-rail code)* este {01, 10}. O eroare unidirecţională transformă acest cod în {00 sau 11} şi astfel ieşirile *checker*-ului vor indica o eroare. Deci, se poate detecta o defecţiune/eroare prin observarea unei ieşiri ilegale la un *checker*, iar pentru a detecta o defecţiune în interior este nevoie de aplicarea unor intrări specifice (configuraţii de biţi, vectori) numite intrări de test. Astfel, se presupune, în mod implicit, că toate intrările de test ale unui sistem TSC sunt aplicate în modul operării normale.

În literatura de specialitate se întâlnesc următoarele definiţii [Gait85], [Lala85], [NaKa88], [John89], [BuJh94]:

- Un circuit se numeşte *fault-secure* (FS) pentru un set Φ de defecţiuni, dacă, pentru orice defecţiune singulară $\tau \in \Phi$, circuitul produce fie o ieşire de cod, aşteptată pentru intrarea respectivă, fie o ieşire ilegală *(noncode)*, dar nu produce un cuvânt de cod neaşteptat pentru acea intrare;
- Un circuit este numit *self-testing* (ST) pentru un set Φ de defecţiuni dacă şi numai dacă, pentru orice defecţiune singulară $\tau \in \Phi$, circuitul produce o ieşire ilegală *(noncode)* pentru cel puţin o intrare din cod;
- Un circuit este numit *totally-self-checking* (TSC) dacă este atât *self-testing*, cât şi *fault-secure* pentru orice defecţiune singulară $\tau \in \Phi$;
- Un circuit se defineşte drept *code-disjoint* (CD) dacă în absenţa defectelor, aplicând intrărilor primare un vector binar şi valid al codului, la ieşire se obţine, de asemenea, un vector binar valid al respectivului cod, aşa cum, dacă aplicând intrărilor primare un vector

binar ilegal pentru codul dat, la ieşire se obţine, de asemenea, un vector binar ilegal pentru al respectivului cod. Un circuit TSC este numit TSC *checker* dacă este *code-disjoint*.

Notă: Pentru circuitele de verificare nu este esenţială proprietatea de *fault-secure*.

Iniţial, definiţia circuitelor TSC a fost dată pentru circuitele combinaţionale, extinzându-se, ulterior, atât asupra circuitelor secvenţiale [Gait85], cât şi, în general, asupra unor sisteme întregi (despre care se poate spune că se comportă ca şi maşinile secvenţiale).

Revenind la *checker*-ul TSC, se punctează faptul că ieşirea lui devine ŞŞ sau 11, dacă şi numai dacă o eroare este detectată la ieşirea unei unităţi funcţionale sau dacă *checker*-ul are o defecţiune *hardware*. Deci, un *checker* TSC trebuie să fie *code-disjoint*.

Adesea există situaţii în care, chiar dacă un *checker* este proiectat pentru *code-disjointness*, nu sunt disponibile de la unitatea funcţională TSC toate intrările de test, necesare pentru testarea lui. Pentru a face un circuit atât *self-testing*, cât şi *code-disjoint* ar trebui să se suspende, din când în când, operaţia normală pentru a alimenta *(to flush) checker*-ul cu intrări de test care în mod normal nu sunt disponibile. Dar, în această situaţie sistemul nu mai este *on line* şi există dificultăţi practice în alimentarea *checker*-ului. O cale mai bună este proiectarea unor *checker*-e care să fie atât *self-testing*, cât şi *code-disjoint* şi care să nu aibă nevoie de acces din exterior, fiind uşor de încorporat.

În conformitate cu cele discutate anterior, un circuit TSC, ale cărui ieşiri sunt codificate într-un cod detector de erori, produce întotdeauna un cuvânt ilegal ca primă ieşire eronată din cauza unei defecţiuni. Acest comportament este foarte avantajos şi este cunoscut sub denumirea de obiectivul proiectării TSC *(TSC goal)*. Există, totuşi, şi alte clase de circuite logice care îndeplinesc acest obiectiv.

Proprietatea de *strongly fault-secure* caracterizează o clasă mai largă de circuite logice care realizează acest obiectiv sub aceleaşi premize privitoare la defecţiunile care se iau în consideraţie. Proprietatea se defineşte în felul următor: un circuit (sistem) este *strongly fault-secure* (SFS) pentru un set Φ de defecţiuni, dacă pentru orice defecţiune singulară $\tau \in \Phi$, circuitul poate fi:

- TSC pentru $\{\tau\}$ sau
- FS pentru $\{\tau\}$ şi, dacă defectul τ apare, atunci circuitul (sistemul) rezultant este tot SFS pentru $\Phi - \{\tau\}$.

Din punct de vedere al proiectantului, este mai avantajoasă proiectarea circuitelor SFS care îndeplinesc obiectivul TSC spre deosebire de proiectarea circuitelor TSC, pentru că cele din urmă impun proiectării mai multe restricţii. De asemenea, chiar dacă proiectarea sistemelor *fail-safe* a fost studiată înainte şi independent de cea a circuitelor TSC şi SFS, circuitele *fail-safe* sunt echivalente cu circuitele *fault-secure* cu o anumită codificare.

O altă proprietate interesantă se defineşte după cum urmează: un circuit este *strongly code-disjoint* (SCD) pentru un set Φ de defecţiuni, dacă pentru orice defecţiune singulară $\tau \in \Phi$, circuitul este fie:

- CD şi este ST pentru $\{\tau\}$, fie
- CD şi, dacă defectul τ apare, atunci circuitul rezultant este tot SCD pentru Φ-$\{\tau\}$.

Combinaţia dintre un circuit TSC (care realizează o anumită funcţie) şi un *checker* TSC aparţine clasei reţelelor logice cu componente TSC. Aceste reţele îndeplinesc obiectivul TSC sub presupunerea defecţiunilor singulare, ceea ce înseamnă că defecţiunile apar una câte una, iar între două defecţiuni este timp suficient astfel încât toate intrările din cod să fie aplicate reţelei. Presupunând că există defecţiuni duble, reţelele de mai sus nu îndeplinesc obiectivul TSC. Prin defecţiuni duble se înţelege că între apariţia a două defecţiuni care afectează circuitul care realizează funcţia dorită sau între apariţia a două defecţiuni care afectează *checker*-ul, există suficient timp pentru ca toate intrările din cod să fie aplicate reţelei. În [Gait88c], Gaitanis prezintă o nouă clasă de circuite de verificare TSC cu proprietăţi mai puternice de autoverificare *(self-checking)* pentru a îndeplini obiectivul TSC în cazul reţelelor de *checker*-e, când se iau în consideraţie defecţiunile duble.

Un TSC *checker* este *strongly-self-checking* (SSC) pentru un set Φ de defecţiuni, dacă *Fc*, *Fe* şi *E* sunt submulţimi disjuncte ale mulţimii de ieşiri ilegale [Gait88c], unde:

- *Fc* este mulţimea ieşirilor neaşteptate, cauzate de defecţiuni interne $\tau \in \Phi$ şi intrări din cod ale *checker*-ului TSC;
- *Fe* este mulţimea ieşirilor neaşteptate, cauzate de defecţiuni interne $\tau \in \Phi$ şi intrări (ilegale) din afara codului ale *checker*-ului TSC;
- *E* este mulţimea ieşirilor produse de *checker*-ul TSC fără defecţiuni cauzate de intrările ilegale.

1.4 Concluzii

Pregătind aplicarea unor metode care să asigure implementarea eficientă a toleranţei la defectare în subsistemele I/O, în acest prim capitol de consistenţă al lucrării au fost supuse unei analize sistematice aspectele de defectare specifice domeniului abordat care să constituie obiective de combătut prin metodele care vor fi dezvoltate în următoarele capitole.

Rezumând, în acest capitol sunt cuprinse următoarele contribuţii:

1) un aport în sistematizare metodelor de creştere a fiabilităţii şi disponibilităţii prin autocontrol implementat atât *hardware*, cât şi *software*

2) definirea unor noi indicatori de fiabilitate specifici matricelor de discuri, necesari pentru evaluarea structurilor de tip RAID şi RAIC, în special RAID7 şi RAIC 7.

2 ANALIZA SUBSISTEMULUI DE DISCURI DIN PUNCT DE VEDERE AL FIABILITĂŢII

În multe sisteme de calcul, subsistemul I/O reprezintă adesea o pondere importantă în cadrul performanţelor sistemului şi din punct de vedere al fiabilităţii reprezintă zona cea mai vulnerabilă, defectarea acestuia putând avea efecte dezastruoase prin pierderea definitivă a datelor.

Dintre componentele subsistemului I/O, discurile magnetice prezintă cel mai ridicat grad de risc în utilizare (componente mecanice în mişcare, etc.).

De asemenea, tot ele sunt cele care au o importanţă decisivă, în performanţă globală a unui sistem.

Creşterea performantelor şi a fiabilităţii discurilor este limitată tehnologic, această limită neputând fi depăşită decât prin utilizarea redundanţei informaţionale şi structurale.

Pornind de la metodele generale de creştere a fiabilităţii sistemelor de calcul, în acest capitol sunt analizate metodele particulare dedicate discurilor cum ar fi: coduri specifice detectoare şi corectoare de erori, precum şi structuri redundante de discuri RAID. În final, se prezintă o clasificare, din punct de vedere al fiabilităţii, pentru structurile redundante de discuri.

2.1 Coduri pentru discuri

De mai mulţi ani, discurile magnetice joacă un rol important în memorarea rapidă şi de mare capacitate a fişierelor în sistemele computerizate. Încă din anii '60, tehnologia discurilor magnetice s-a îmbunătăţit continuu din punct de vedere al densităţii, capacităţii de stocare, vitezei de transfer, costului şi fiabilităţii. De exemplu, în ceea ce priveşte densitatea spaţială de memorare, au avut loc îmbunătăţiri de patru ordine de mărime — de la 10^3 la 10^7 biţi pe ţol pătrat.

Viteza de transfer a datelor este mai mare în cazul discurilor magnetice, decât în cazul benzilor magnetice, în special datorită vitezei mai mari de rotaţie. De aceea, logica sa trebuie să admită o mai mare toleranţă. În etapele incipiente ale dezvoltării discurilor se foloseau doar verificările simple ale parităţii pentru a testa integritatea datelor. În etapa următoare s-a făcut uz de coduri polinomiale pe 16 biţi pentru detectarea erorilor salvă. În anii '70 discurile de înaltă densitate şi de mare viteză (de exemplu, IBM 3330) au pus bazele unei noi tehnologii. Acest tip de disc

folosea codurile *Fire* pentru a avea capacitatea detecţiei şi corecţiei erorilor salvă. În proiectele recente, acest tip de cod este înlocuit cu un cod întreţesut *Reed-Solomon* pentru corecţia erorilor. Pe lângă codurile detectoare şi corectoare de erori se mai folosesc şi alte tehnici recuperatorii pentru a mări fiabilitatea şi a creşte securitatea informaţiilor, cum ar fi: ignorarea defectelor, blocuri alternative de date şi recitirea .

În sistemele de discuri magnetice predomină erorile de tip salvă. Majoritatea acestora este legată de imperfecţiunile suprafeţei discurilor sau de neregularităţile suprafeţei acestora. Restul erorilor este, de regulă, datorat capetelor magnetice, susceptibile de erori aleatoare induse de zgomote [Howe84].

Discurile optice reprezintă o tehnologie relativ nouă pentru înregistrarea informaţiilor. Fiecare disc este acoperit cu o peliculă reflectorizantă. Citirea şi scrierea sunt realizate prin intermediul unui fascicul laser. Mai precis, citirea se reduce la a stabili reflectivitatea unei anumite zone a discului, în vreme ce la scriere un laser de putere mai mare realizează, prin topire, adâncituri în pelicula reflectorizantă. Densitatea informaţiilor pe un disc optic este mult mai mare decât în cazul discurilor magnetice, de vreme ce tehnologiile optice permit o „rezoluţie" mai mare. De aceea, se estimează că acest mediu de stocare va avea un rol tot mai important în sistemele de memorare.

Erorile sunt datorate, în principal, discului optic însuşi, de vreme ce nu se pot încă fabrica la un preţ rezonabil pelicule reflectorizante de înaltă stabilitate şi înaltă fiabilitate. Aproape toate erorile au drept cauză imperfecţiunile discului, iar restul sunt datorate variaţiilor de focalizare la înregistrare sau zgomotului aleator la citire. Ca atare, sunt necesare atât facilităţi pentru corecţia erorilor salvă, cât şi pentru corecţia celor aleatoare [SaTN86, IYHI87]. În prezent se folosesc tehnici puternice de corecţie a erorilor, cum ar fi codul dublu codificat *Reed-Solomon* sau o combinaţie a codului *Reed-Solomon* cu un cod ciclic, pentru a obţine acelaşi nivel de fiabilitate ca în cazul discurilor magnetice.

2.1.1 Codul *Fire* pentru memoriile bazate pe disc magnetic

Codul *Fire* [Fire59], care este capabil de corecţia oricărei salve de lungime l sau mai mică şi de detectarea simultană a oricărei salve de lungime $d \geq l$, este generat de următorul polinom generator:

$$g(x) = (x^{2l-1} + 1)\ p(x)$$

Aici $p(x)$ este un polinom ireductibil de grad m cu exponentul e şi $c \geq l + d - 1$. Lungimea n a acestui cod este egală cu cel mai mic multiplu comun (CMMMC) dintre e şi c:

$$n = \text{CMMMC}\ (e, c)$$

Numărul biţilor de verificare a acestui cod este $(c + m)$.

Fie $b(x)$ polinomul erorii salvă, al cărui grad este mai mic sau egal cu $l - 1$. Atunci $r(x)$ are expresia:

$$r(x) = v(x) + x^i \bullet b(x)$$

unde i arată poziţia de începere a erorii salvă. Dacă $b(x) = 0$, atunci $r(x)$ poate fi divizat atât de $(x^c + 1)$, cât şi de $p(x)$, reziduul fiind, deci, egal cu zero. Dacă $b(x) \neq 0$, atunci registrele de deplasare vor conţine următoarele reziduuri:

$$x^i \bullet b(x) = s_1(x) \mod (x^c + 1) \quad (1)$$

$$x^i \bullet b(x) = s_2(x) \mod p(x) \quad (2)$$

Folosind aceste reziduuri, $s_1(x)$ şi $s_2(x)$, trebuie să obţinem $b(x)$ şi valoarea lui i. După $(n - 1)$ deplasări ale registrelor de deplasare, acestea vor conţine următoarele reziduuri:

$$\begin{aligned} s_1(x) \bullet x^{n-1} &= x^n \bullet b(x) \\ &= b(x) \mod (x^c + 1) \\ s_2(x) \bullet x^{n-1} &= x^n \bullet b(x) \\ &= b(x) \mod p(x) \end{aligned}$$

Cu alte cuvinte, atunci când conţinutul părţii inferioare l a registrului tipului de eroare este egal cu cel al registrului localizării erorii după δ deplasări ale conţinutului acestor registre, tipul erorii salvă, exprimat sub forma lui $b(x)$, poate fi introdus în aceste registre. Pe lângă acest polinom $b(x)$, poziţia i a erorii se poate calcula din δ (care este egal cu $n - i$), ffind, deci, $i = n - \delta$.

Această metodă de decodificare necesită un timp mai mare atunci când i este mic în comparaţie cu n. De aceea, vom lua în consideraţie metoda decodificării de înaltă viteză [Chie69]. Să presupunem că c şi e sunt prime unul faţă de altul.

În această metodă de decodificare, cuvântul recepţionat $r(x)$ este introdus simultan în registrul tampon, registrul tipului de eroare şi

registrul localizării erorii. Apoi, se obţin reziduurile exprimate în ecuaţiile 1 şi 2 în registrul tipului erorii şi, respectiv, în registrul localizării erorii. În acest moment, putem deduce cuvântul recepţionat, pe baza reziduurilor astfel obţinute, astfel:

$$s_1(x) = s_2(x) = 0 \rightarrow r(x) : \textit{fara eroare}$$

$$\left.\begin{array}{l} s_1(x) \neq 0 \text{ and } s_2(x) = 0 \\ s_1(x) = 0 \text{ and } s_2(x) \neq 0 \end{array}\right\} \rightarrow r(x) : \textit{eroare} \cdot \textit{necorectabila}$$

Dacă $s_1(x) \neq 0$ şi $s_2(x) \neq 0$, atunci decodificarea se poate face conform următorului algoritm:

1) Deplasează registrul tipului erorii până când în registrul părţii superioare $(c - 1)$ sunt doar zerouri şi memorează numărul de deplasări δ_0. Dacă respectivul conţinut este diferit de zero după $(c - 1)$ deplasări, atunci a apărut o eroare necorectabilă şi, deci, decodificarea se opreşte;

2) Deplasează registrul localizării erorii până când conţinutul său este egal cu partea inferioară 1 a registrului tipului de eroare şi memorează numărul de deplasări δ_1. Dacă valoarea părţii inferioare a registrului tipului de eroare nu este egală cu cea a registrului localizării erorii după $(e - 1)$ deplasări, atunci se presupune că a apărut o eroare necorectabilă şi, deci, procesul va fi oprit;

3) Poziţia i a erorii se poate calcula din valorile δ_0 şi δ_1, eroarea corectându-se, deci, prin folosirea tipului de eroare $b(x)$. Cu alte cuvinte, conţinutul părţii inferioare 1 a registrului tipului de eroare denotă polinomul $b(x)$.

În algoritmul de mai sus, poziţia i a erorii se poate calcula folosind Teorema Restului Chinezesc, astfel încât

$$\begin{aligned} \delta_0 &= c - i \\ &= -i \bmod c \\ \delta_1 &= e - i \\ &= -i \bmod e \end{aligned}$$

după care $-i = \delta_0 \cdot A_0 \cdot e + \delta_1 \cdot A_1 \cdot c \bmod n$, unde A_0 şi A_1 sunt întregi care satisfac relaţia:

$$A_0 \cdot e + A_1 \cdot c = 1 \bmod n$$

Calcularea lui A_0 şi A_1 se poate face cu uşurinţă *off-line*, cu numerele $A_0 \cdot e$ şi $A_1 \cdot c$ memorate în calculatorul poziţiei erorii. În cazul acestei decodificări, numărul maxim de deplasări este $(e + c - 2)$, în timp ce la

decodicarea anterioară era $(e \cdot c - 1)$, ceea ce face acest decodificator mult mai rapid decât cel precedent.

În continuare, ne vom concentra asupra unei noi clase de coduri care este o generalizare a codurilor *Fire* [Chie69]. Acestea sunt generate de un polinom de forma

$$g(x) = (x^c - 1) \prod_{i=1}^{h} p_i(x) \tag{3}$$

unde $p_i(x)$ este un polinom ireductibil distinct, având exponentul e_i şi gradul r_i. Se mai presupune că exponenţii e_i, unde $i = 1, 2, \ldots, h$, nu divid c. Lungimea codului este

$$n = \text{CMMMC}\ (c, e_1, e_2, \ldots, e_h)$$

Teorema 1 [Chie69]. *Codul generat de $g(x)$, dat de ecuaţia 3 detectează toate erorile salvă, mai puţin cele cu lungimea mai mică sau egală cu d; corectează toate salvele $b(x)$ de lungime mai mică sau egală cu l, care sunt prime în raport cu $\prod_{i=1}^{h} p_i(x)$, cu condiţia ca*

$$c \geq l + d - 1 \text{ şi } l \leq \sum_{i=1}^{h} r_i.$$

Ecuaţiile care determină poziţia i pot fi scrise astfel:

$$-i = \delta_0 \cdot A_0 \cdot \prod_{j=1}^{h} e_j + c \cdot \sum_{j=1} \delta_j \cdot A_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{h} e_k \tag{4}$$

$$A_0 \cdot \prod_{j=1}^{h} e_j + c \cdot \sum_{j=1}^{h} A_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{h} e_k = 1 \tag{5}$$

Sistemele de discuri compatibile cu IBM 3330 folosesc codul *Fire* generalizat, generat cu ajutorul polinomului următor:

$$g(x) = (x^{22} + 1) \prod_{i=1}^{3} p_i(x)$$

unde

$$p_1(x) = 1 + x + x^6 + x^7 + x^{11}$$

$$p_2(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{12}$$

$$p_3(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11}$$

sunt polinoame ireductibile distincte. Exponenţii polinoamelor $p_1(x)$, $p_2(x)$ şi $p_3(x)$ sunt 89, 13 şi, respectiv, 23. Lungimea codului este, deci:

$$n = \text{CMMMC}\ (22, 89, 13, 23) = 585\,442$$

El are $22 + 11 + 12 + 11 = 56$ biţi de verificare şi este capabil să corecteze orice salvă de lungime 11 sau mai mică.

Din ecuaţiile (4) şi (5) se pot deduce cele care determină poziţia erorii după cum urmează:

$$i = \delta_0\ A_0 e_1 e_2 e_3 + \delta_1\ A_1 c\ \ e_2 e_3 + \delta_2\ A_2 c\ \ e_1 e_3 + \delta_3\ A_3 c\ \ e_1 e_2 \qquad (6)$$

unde A_0, A_1, A_2 şi A_3 satisfac relaţia:

$$A_0 e_1 e_2 e_3 + A_1 c\ \ e_2 e_3 + A_2 c\ \ e_1 e_3 + A_3 c\ \ e_1 e_2 = -1 \bmod n \qquad (7)$$

Pentru acest cod putem afla că:

$$\begin{aligned} A_0 e_1 e_2 e_3 &= -1 \bmod c \\ A_1 c\ \ e_2 e_3 &= -1 \bmod e_1 \\ A_2 c\ \ e_1 e_3 &= -1 \bmod e_2 \\ A_3 c\ \ e_1 e_2 &= -1 \bmod e_3 \end{aligned} \qquad (8)$$

Cei mai mici întregi care satisfac ecuaţiile (7) şi (8) sunt: $A_0 = 5$, $A_1 = 78$, $A_2 = 6$ şi $A_3 = 10$. Aceşti întregi se introduc în ecuaţia (6), rezultând următoarea ecuaţie care ne indică poziţia erorii:

$$i = 133\,055\ \delta_0 + 513\,048\ \delta_1 + 270\,204\ \delta_2 + 254\,540\ \delta_3 \bmod 585\,442$$

Operaţiile de deplasare ale registrului tipului de eroare se execută până când cele mai semnificative 10 ranguri ale acestuia au toate valoarea 0. Deplasarea registrelor de localizare a erorii se execută până când întregul conţinut al tuturor celor trei registre coincide cu cele mai puţin semnificative 11 ranguri ale registrului tipului de eroare. Considerând că numărul acestor deplasări este $\delta_0, \delta_1, \delta_2$ şi, respectiv, δ_3, rezultă $K_0 = 133\,055\ \delta_0$, $K_1 = 513\,048\ \delta_1$, $K_2 = 270\,204\ \delta_2$ şi, respectiv, $K_3 = 254\,540\ \delta_3$. Putem, acum, obţine i ca fiind $i = K_0 + K_1 + K_2 + K_3 \bmod n$.

O altă metodă rapidă de corecţie, bazată pe codul *Fire* obişnuit, a fost propusă în [Adi84]. Aceasta este o tehnică hibridă care combină capturarea erorilor secvenţiale şi tehnicile de căutare în tabele. Decodificatorul necesită cel mult $(c\ \ m - 1)$ cicluri de deplasare pentru a găsi atât eroarea de tip salvă, cât şi locaţia ei. Implementarea decodificatorului necesită memorii programabile numai de citire

(*PROM*-uri) şi matrice logice programabile (*FPGA*-uri), potrivite implementărilor pe scară largă.

2.1.2 Codul întreţesut R-S SbEC-DbEC pentru memoriile cu discuri magnetice

O schemă de corecţie a erorilor de tip salvă, de înaltă performanţă, care foloseşte un cod corector de eroare pe *byte* este dat [Hori75b]. Discurile magnetice apărute mai recent, cum ar fi cele din sistemele IBM 3370 şi IBM 3380, adoptă un cod de distanţă 4, *Reed-Solomon* (R-S), discutat în [Hodg80]. Acest sistem utilizează un format fix al blocului de date. Fiecare bloc are 512 *bytes*. Pentru controlul erorilor, acestuia i se mai adaugă încă nouă biţi. Codul folosit în acest sistem este o prescurtatre a codului *Reed-Solomon* de distanţă 4, cu simboluri din câmpul $GF(2^8)$. Polinomul generator al codului este:

$$g(x) = (x + 1)(x + \alpha)(x + \alpha^{-1})$$

unde α este elementul primitiv al lui $GF(2^8)$ şi rădăcină a lui $p(x) = 1 + x^2 + x^3 + x^4 + x^8$. Acest cod este capabil să corecteze orice eroare monosimbolică (un *byte*) şi să detecteze orice eroare bisimbolică (doi *bytes*).

Pentru codificare, fiecare bloc este subîmpărţit în trei subblocuri: D_{-1}, D_0 şi D_1, fiecare având 171 de *bytes*. Cele trei blocuri se reprezintă în formă polinomială după cum urmează:

$$D_i = D_{i,170}\, x^{170} + D_{i,169}\, x^{169} + \ldots + D_{i,1}\, x + D_{i,0}$$

$$i = \{-1, 0, 1\}$$

unde fiecare *byte* $D_{i,k}, i = \{-1, 0, 1\}, k = \{0, 1, \ldots, 170\}$, este privit ca un simbol în $GF(2^8)$. Cei trei *bytes* de verificare pentru D_i, notaţi C_{-1}, C_0, C_1, unde $i = \{-1, 0, 1\}$, se obţin ca mai jos:

$$C_{i,j} = x \cdot D_i(x) \bmod (x + \alpha^j)$$

$$i = \{-1, 0, 1\} \quad j = \{-1, 0, 1\}$$

A se observa diferenţa dintre această metodă de codificare şi cea folosită în cazul codurilor ciclice, unde toţi cei trei *bytes* sunt calculaţi prin împărţirea lui $x^3 \cdot D_i(x)$ la $g(x)$. După ce s-au format cei trei *bytes* de control, aceştia sunt adăugaţi la D_i pentru a forma un cuvânt codificat W_i.

Cele trei cuvinte codificate W_1, W_0 şi W_{-1} sunt întreţesute pentru a forma un bloc codificat. Ca urmare, codul general rezultat este (174,171) R-S, întreţesut de gradul 3. Acest cod este capabil să corecteze orice salvă de erori de cel mult trei *bytes* şi să detecteze salve eronate de cel mult şase *bytes*.

Atunci când un bloc codificat este citit de pe disc, el este descompus în trei subcuvinte $W_{i'}$, $W_{0'}$ şi $W_{-1'}$, unde:

$$W_{i'} = (C_{i',1} , C_{i',0} , C_{i',-1} , D_{i'})$$

$$i = \{ 1 , 0 , -1 \}$$

Sindromul $S_{i,j}$ ($i = \{ 1 , 0 , -1 \}$, $j = \{ 1 , 0 , -1 \}$) se poate obţine în felul următor:

$$S_{i,j} = C'_{i,j} + x \cdot D'_i(x) = C'_{i,j} + \alpha \cdot D'_i(\alpha) \quad mod\ (x+\alpha^j)$$

unde $D_{i'}(x) = D_{i',170}\, x^{170} + \ldots + D_{i',1}\, x + D_{i',0}$. Dacă $S_{i,1} = S_{i,0} = S_{i,-1} = 0$, cuvântul nu are erori. Dacă toate sindroamele $S_{i,1}$, $S_{i,0}$ şi $S_{i,-1}$ sunt diferite de zero, atunci există erori de un singur *byte* în cuvântul citit. Acest lucru se datorează faptului că, dacă o eroare α^m $(0 \le m \le 254)$ este adăugată simbolului de date $D_{i,l}$ $(l = \{ 0,1,\ldots,170) \}$, atunci expresia cuvântului citit este

$$\begin{aligned} D_{i',l} &= D_{i,l} + \alpha^m \\ D_{i',k} &= D_{i,k} \quad k \neq l \,, k = \{ 0 , 1 , \ldots , 170 \} \\ C_{i',j} &= C_{i,j} \end{aligned}$$

şi, deci, sindroamele nu toate egale cu zero se pot obţine din:

$$\begin{aligned} S_{i,1} &= \alpha^{m \cdot l} \\ S_{i,0} &= \alpha^{m} \\ S_{i,-1} &= \alpha^{m \cdot (l^{-1})} \end{aligned}$$

În acest caz, tipul erorii poate fi exprimat ca $S_{i,0}$, iar poziţia erorii se exprimă cu ajutorul următoarelor relaţii:

$$\frac{S_{i,1}}{S_{i,0}} = \frac{\alpha^{m \cdot l}}{\alpha^{m}} = \alpha^{l-1}$$

$$\frac{S_{i,0}}{S_{i,-1}} = \frac{\alpha^{m}}{\alpha^{m \cdot (l^{-1})}} = \alpha^{l-1}$$

2.1.3 Codul R-S întreţesut-încrucişat pentru discuri optice

După cum s-a arătat mai înainte, erorile din sistemele optice sunt aproape întotdeauna datorate suprafeţei discului şi, de aceea, în acest caz avem de a face nu numai cu erori de tip salvă, ci şi cu erori aleatoare. De aceea avem nevoie de un cod pentru corecţia ambelor tipuri de erori.

Codul R-S întreţesut-încrucişat pentru discuri optice *(CIRC — Cross Interleaved Reed-Solomon Code)* face parte dintr-o nouă clasă de coduri dublu codificate, în care cel de al doilea cod *Reed-Solomon* codifică ieşirea întârziată şi dispersată (de exemplu, *întreţesut-încrucişat*) a primului cod codificat *Reed-Solomon* . Pentru a ilustra conceptul acestei clase de coduri este oferit un mic exemplu.

Exemplu

Fie două coduri C_1 şi C_2, capabile să corecteze, fiecare în parte, erori singulare. Să presupunem că simbolurile de verificare P şi Q sunt reprezentate prin două simboluri, pentru C_1 şi, respectiv, C_2.

Simbolurile de verificare P_{4n}, sunt obţinute din simbolurile consecutive: D_{4n}, D_{4n-1}, D_{4n-2} şi D_{4n-3}, obţinute prin folosirea unui codificator C_2. Simbolurile codificate D_{4n}, D_{4n-1}, D_{4n-2}, D_{4n-3} şi P_{4n} sunt întârziate cu lungimi inegale, devenind D_{4n}, D_{4n-3}, D_{4n-6}, D_{4n-9} şi P_{4n-16}. Întârzierile sunt diferite pentru fiecare dintre simboluri. Simbolurile de verificare Q_{4n} sunt produse pe baza simbolurilor întârziate, de către un codificator C_1. Ca rezultat al întreţeserii încrucişate, chiar dacă apar erori consecutive de patru simboluri, de exemplu, erori în D_7, D_{10}, D_{13} şi D_{16}, care nu pot fi corectate în C_1, acestea sunt dispersate ca erori singulare în fiecare cod C_2, putând fi, deci, corectate de către acest decodificator. Dacă apar erori în trei simboluri, de exemplu în D_{13}, D_{16} şi D_{17}, acestea pot fi, de asemenea, corectate în paşi secvenţiali, astfel: eroarea singulară din D_{13} poate fi corectată în decodificarea C_2, după care eroarea singulară din D_{16} este corectată în C_1 şi, în final, eroarea singulară din D_{17} se corectează în C_2. Totuşi, se poate arata că erorile din patru simboluri, de exemplu în D_{13}, D_{14}, D_{16} şi D_{17}, nu pot fi corectate, de vreme ce sunt câte două erori în sarcina decodoarelor C_1 şi C_2, încă din prima etapă. Pentru a corecta în mod adecvat aceste patru erori se apelează la tehnica de corecţie a ştergerii, care permite corecţia erorilor duble prin folosirea pointerilor de eroare pentru codurile corectoare a erorilor de simbol. De

aceea, în prima etapă, când erori de două simboluri sunt detectate în timpul decodificării C_1, (de exemplu, în acest caz codul C_1 joacă rolul detectorului de eroare dublă), sunt iniţializaţi pointeri de eroare pentru toate simbolurile de pe rampa crescătoare. În etapa următoare, decodificarea C_2 poate corecta aceste erori duble, folosind respectivii pointeri. Acest lucru este posibil datorită faptului că amintitele coduri corectoare C_1 şi C_2 sunt coduri de distanţă 3, putând, deci, să corecteze ştergeri cel mult duble.

În cazul sistemelor de înregistrare a compact-discurilor audio se folosesc următoarele două coduri corectoare de erori *Reed-Solomon*, de distanţă 5, peste $GF(2^8)$ [VINH80]. Codurile C_1 şi C_2 sunt după cum urmează:

C_1: (32,28) cod corector *Reed-Solomon* de eroare dublă;

C_2: (28,24) cod corector *Reed-Solomon* de eroare dublă.

În această schemă de codificare se pot lua în consideraţie multe strategii. Printre acestea, sistemele de tip compact-disc adoptă următoarea strategie, numită *super-strategie*:

1) Decodificarea C_1

- Sindromul „totul pe zero“ → nu sunt erori;
- Erori de un singur simbol → corecţie a erorii singulare;
- Erori de două simboluri → corecţie a erorii duble. Pointerii de localizare a erorii sunt transmişi la decodificatorul C_2;
- Erori de mai mult de trei simboluri → doar detectarea erorilor. Pointerii de localizare a erorilor sunt transmişi la decodificatorul C_2;

2) Decodificarea C_2:

- Sindromul „totul pe zero“→ nu sunt erori;
- Erori de un singur simbol → corecţie a erorii singulare;
- Erori de un singur simbol →
  - ♦ Numărul pointerilor erorilor de la decodificatorul C_1 este mai mic sau egal cu 4 şi, în acelaşi timp, localizarea erorilor a doi dintre poinerii de eroare este egală cu cea calculată de către sindrom → corecţia erorii duble;
  - ♦ Altele → doar detecţia erorilor;

- (iv) Erori de mai mult de trei simboluri → doar detectarea erorilor.

A se observa că, atunci când erorile duble sunt corectate în timpul decodificării C_1, aceşti doi pointeri de localizare a erorilor sunt transmişi decodificării C_2 corectarea greşită a erorilor triple, confundate ca erori duble în timpul decodificării C_2. În cazul în care nici decodificatorul C_2 nu poate corecta erorile, el lasă să treacă 24 de simboluri necorectate de date, dar marcate cu pointerii de eroare proveniţi de la C_1. În acest fel chiar dacă decodificatorul C_2 nu poate decodifica, majoritatea simbolurilor sunt foarte lipsite de erori, iar valorile eşantioanelor marcate şi necorectate pot fi refăcute pe baza unei interpolări lineare [VINH80] în sistemele audio digitale. În cazul unui eşantion marcat ca eronat, valoarea acestuia se poate deduce din cea a eşantioanelor corecte care îl mărginesc şi care nu au fost marcate cu pointeri de eroare. Această schemă *CIRC* cu super-strategie are o capacitate maximă de corecţie completă a salvelor de erori de până la 4000 de biţi.

Pentru sistemele de stocare digitală, numite *CD-ROM*, interpolarea lineară nu constituie o soluţie. De aceea trebuie adăugate alte metode de creştere a calităţii datelor. Un sistem adoptă atât codurile dublu codificate *R-S SbEC*, cât şi o verificare ciclică a redundanţei *(CRC)* pentru datele deja cuprinse în *CIRC*. Cu alte cuvinte, *CD-ROM* foloseşte o metodă atât de puternică pentru corecţia erorilor, încât biţii *CRC* sunt adăugaţi la codificarea *CIRC*, rezultând informaţie dublu codificată în codurile *R-S SbEC*. Astfel, datele sunt cvadruplu codificate, iar dacă se include şi *CRC*, atunci sunt cvintuplu codificate. Un disc are o capacitate de circa 650 MB.

1) Codurile cu verificare ciclică a redundanţei *(CRC).* Acest cod a fost adăugat pentru a verifica dacă nu a apărut o corectare eronată a codurilor *Reed-Solomon* sau pentru a detecta erori necorectabile. Polinomul generator al codului *CRC* este dat mai jos:

$$g(x) = (x^{16} + x^{15} + x^2 + 1)(x^{16} + x^2 + x + 1)$$

`2) Codurile dublu codificate *Reed-Solomon.* Aceste coduri sunt determinate de către următorul polinom primitiv $g'(x)$, iar α este un element prim în $GF(2^n)$ (de exemplu, o rădăcină a lui $g'(x)$).

$$g'(x) = x^8 + x^4 + x^3 + x^2 + 1$$

Codul include două coduri *Reed-Solomon* (de exemplu, codul C_1 (26,24) *R-S* şi codul C_2 (45,43) *R-S*). Ambele coduri sunt coduri *Reed-Solomon* cu distanţa 3.

Octeţii de verificare pentru C_1 sunt generaţi din informaţiile incluse în direcţia verticală. Octeţii de verificare pentru C_2 sunt generaţi din informaţiile conţinute pe diagonala descrescătoare din zona datelor utile (care are dimensiunea de 24H43 de octeţi, adică 1032 de octeţi, în format bidimensional). Matricele H ale codurilor C_1 şi C_2 sunt după cum urmează:

$$H_1 = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ \alpha^{25} & \alpha^{24} & \alpha^{23} & \dots & \alpha & 1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & & 1 & 1 \\ \alpha^{44} & \alpha^{43} & \alpha^{42} & \dots & \alpha & 1 \end{bmatrix}$$

Întâi se generează codul C_1, apoi codul C_2. De observat este că decodificarea este permisă pentru oricare dintre secvenţe, adică din C_1 în C_2 şi invers. În plus aceste secvenţe pot fi şi repetitive, ca de exemplu: decodificare $C_1 \rightarrow$ decodificare $C_2 \rightarrow$ decodificare $C_1 \mapsto$ decodificare C_2 ... Mai mult, putem folosi rezultatele decodificării *CIRC*. Fiecare dintre coduri are următoarele posibilităţi de corecţie a erorilor:

® Posibilitatea corecţiei erorilor de un singur octet;

© Posibilitatea corecţiei ştergerilor de doi octeţi.

Ca atare, există multe strategii posibile de decodificare. De exemplu, folosind capacitatea de corecţie a ştergerilor, marcată cu ©, se poate aplica următorul model:

$$C_1© \rightarrow C_2© \rightarrow C_1© \rightarrow C_2© \rightarrow \dots$$

Pentru a ne proteja împotriva corecţiilor greşite din ©, codul *CRC* verifică aceste corecturi greşite şi identifică erorile necorectabile. O astfel de strategie de decodificare poate atinge aceeaşi rată a erorilor ca unităţile comerciale de discuri magnetice sau de benzi magnetice.

Alt exemplu de *CD-ROM*, cu o capacitate de 2,6 GB, adoptă atât un cod *Reed-Solomon* întreţesut, de distanţă 5, cât şi un cod de verificare ciclică a redundanţei pentru fiecare grupare 512 octeţi de date utile, numită sector.

În acest sistem este adoptat un cod *C Reed-Solomon* (39,35) de distanţă 5 peste $GF(2^8)$, putându-se astfel corecta orice eroare de dublu-octet. Astfel, codul întreţesut folosit *Reed-Solomon* poate corecta până la 30 de octeţi greşiţi într-un sector. În plus, acest sistem adoptă un cod de verificare ciclică a redundanţei (unul cu patru octeţi de verificare), care verifică informaţiile corectate de către codurile *Reed-Solomon* şi oferă protecţie împotriva tentativei de corectare a erorilor necorectabile, care depăşesc posibilitatea de corecţie a codurilor *Reed-Solomon*. Codul de verificare ciclică a redundanţei este adăugat la sfârşitul zonei datelor utile.

Alt tip de sistem de discuri optice, aşa numitul „disc optic cu o scriere şi multiple citiri" *(CD-WORM - rite Once Read Many)*, având un diametru de 5,25 ţoli, este o propunere relativ recentă pentru memoria de masă a calculatoarelor. În cazul acestor discuri, există două tipuri de distribuţii ale erorilor: una este cauzată de o scurtă salvă greşită (mai puţin de zece biţi) cu probabilitate ridicată de apariţie, iar cealaltă de o lungă salvă eronată (între 10 şi 100 de biţi) cu probabilitate scăzută de apariţie [SaTN86]. Următoarele strategii au fost propuse pentru corectarea acestor tipuri de erori: un cod *Reed-Solomon* de mare distanţă ($d \approx 1^-$) cu o lungime a codului de 120 până la 140 de octeţi, întreţesut într-un raport de la 4 până la 10, precum şi un cod produs care foloseşte două coduri *Reed-Solomon* (având distanţa cuprinsă între 3 şi 5), ceea ce ar da naştere unui cod cu o distanţă cuprinsă între 15 şi 25, cu o lungime a codului de la 30 până la 50 de octeţi [Yama86], [Kurt87].

Discurile optice reinscriptibile, dezvoltate recent, funcţionează pe baza unui principiu magneto-optic şi prezintă caracteristici similare ale erorilor apărute. De aceea, codurile *Reed-Solomon* de mare distanţă li se aplică şi lor cu aceeaşi eficienţă [FRSH87, IYHI87, KKIS87].

2.2 Matrice de discuri independente şi redundante

Asigurarea protecţiei datelor faţă de situaţiile anormale, inclusiv erori umane, căderi de echipamente şi condiţii ostile a apărut ca o necesitate stringentă chiar cu primul sistem de calcul. *Backup*-ul, copierea periodică a datelor pe un suport mobil păstrat în afara sistemului, rămâne soluţia esenţială practică, în ciuda abilităţii date de creşterea numărului de discuri utilizate pentru a preveni alterarea accesului la date corecte în situaţii de căderi sau dezastre. Şi mai puţine progrese s-au realizat în domeniul minimizării erorilor umane ceea ce impune utilizarea în continuare a procedurilor de *backup*, chiar şi pentru sistemele de discuri care pot supravieţui la căderi şi dezastre.

Necesitatea ca sistemul de discuri să menţină un acces continuu *on-line* a apărut în anii '70, o dată cu creşterea masivă a utilizării sistemelor de calcul *on-line*. *Backup*-ul şi restaurarea *off-line* erau considerate suficiente pentru a acoperi o cădere. Deoarece accesul la date corupte sau distruse nu are nici o valoare, proiectanţii au fost obligaţi să furnizeze soluţii de protejare a datelor şi accesul la acestea în cazul unor căderi. Deoarece datele sunt memorate, în principal, pe discuri, prima soluţie apărută a constat în memorarea aceloraşi informaţii pe două suporturi diferite *(mirroring)*, metoda care protejează datele şi menţine accesul la ele chiar şi în situaţia căderii unui disc din perechea utilizată.

Soluţia „mirroring" presupune costuri cel puţin duble pentru subsistemul de discuri, faţă de cel clasic, astfel că a apărut necesitatea practică de a găsi soluţii mai puţin costisitoare pentru creşterea protecţiei datelor şi asigurarea accesului continuu la acestea. Astfel, la sfârşitul anilor '80 s-au pus bazele conceptului de RAID – Redundant Array of Independent Disks. RAID utilizează redundanţa sub forma unor informaţii de control (paritate). Faţă de *mirroring*, unde redundanţa era de 100%, la RAID ea poate scădea şi sub 10%.

Pentru o tratare unitară, soluţia *mirroring* a fost inclusă în familia RAID care a fost divizată în două:

- *mirrored* RAID
- *parity* RAID – RAID protejat prin paritate

Spre deosebire de *mirrored* RAID, primele *parity* RAID–uri sufereau de o foarte scăzută performanţă datorată necesităţii generării şi scrierii parităţii în timpul operaţiei de scriere şi a regenerării *on-line* ca răspuns la o cerere către un disc căzut. Această problemă a fost rezolvată parţial prin tehnici precum *caching*, scriere asistată pe discuri etc. şi a condus la o dezvoltare exponenţială a utilizării RAID-ului după mijlocul anilor '90.

Două decade de cercetări în domeniul protecţiei datelor şi menţinerii *on-line* a accesului în cazul unei căderi sau a unui dezastru au generat o largă paletă de soluţii de sisteme de discuri cu diverse grade de rezistenţă. Pentru evitarea confuziilor şi pentru a uşura alegerea celei mai potrivite soluţii pentru o anumită aplicaţie a fost înfiinţată o asociaţie internaţională a cercetătorilor din domeniul RAID la care sau afiliat şi cei mai importanţi producători, asociaţie denumită RAID Advisory Board's (RAB).

Primele mele cercetări în domeniul fiabilităţii transmiterii şi stocării datelor s-au focalizat, începând cu anul 1991, în subdomeniul RAID, în principal abordând problematica legată de „Algoritmii de distribuţie a datelor, parităţii şi spaţiilor de rezervă". În multe sisteme de calcul, subsistemul I/O reprezintă adesea o pondere importantă în cadrul performanţelor sistemului. De-a lungul ultimilor ani, viteza CPU a crescut în proporţii de 40% până la 100% pe an, pe când timpul căutării pe disc s-a îmbunătăţit cu doar 7% pe an (creşteri asemănătoare de performanţă se pot observa şi la celelalte componente ale subsistemului I/O).

Acest fapt a condus la o mare discrepanţă între viteza CPU/memorie principală şi cea a subsistemului operaţiilor I/O, şi este de aşteptat ca aceasta să crească şi mai mult în viitorul apropiat. Bazându-ne pe aceste observaţii, este de prevăzut ca viitoarele creşteri ale vitezei de lucru ale procesoarelor să determine doar o mică îmbunătăţire a performanţelor totale ale sistemului de calcul. Se ridică următoarea problemă fundamentală: bazându-ne pe viitoarele dezvoltări predictibile ale tehnologiei în domeniul discurilor, cum se poate reduce discrepanţa de viteză dintre CPU/memorie principală şi subsistemul I/O?

Evident, datorită limitărilor tehnologice, soluţiile pot fi găsite doar la nivel structural.

O idee atractivă este cea numită matrice de discuri *(disk array)*, unde subsistemul de discuri este compus din mai multe discuri, cu datele împrăştiate pe toate aceste discuri. Conceptele de „disc întreţesut" *(disk interleaving)* şi „disc secţionat" *(disk striping)* au fost pentru prima oară folosite de către Kim [Kim85], respectiv Salem şi Garcia-Molina [GaMo87]. De atunci, o mare cantitate de muncă a fost depusă asupra mai multor modele, referitor la performanţele matricelor de discuri şi a fiabilităţii acestora. Matricele de discuri au fost împărţite în şase clase denumite RAID *(Redundant Arrays of Inexpensive Disks)* [PDGG88]. Printre cele cinci, cele mai promiţătoare două candidate la obţinerea de performanţe ridicate într-un sistem de calcul par a fi RAID 1 (matrice de discuri oglindită - *mirrored disk array*) şi RAID 5 (matrice cu paritate rotită - *rotated parity array*), RAID 6 nefiind încă unanim acceptată drept RAID de sine stătător, iar RAID 7 reprezentând un rezultat al autorului acestei lucrări.

Pe măsură ce performanţa altor componente ale sistemului continuă să se dezvolte rapid, performanţa subsistemului de stocare devine tot mai importantă. Performanţa subsistemului de stocare şi fiabilitatea pot fi crescute prin gruparea logică a mai multor unităţi de disc în matrice de

discuri. Organizările matricelor de discuri sunt definite de schemele de distribuţie şi de mecanismele de redundanţă. Diferitele combinaţii ale acestor două componente fac matricele de discuri potrivite pentru o scară largă de aplicaţii. Multe decizii de implementare a matricelor rezultă, de asemenea, din compromisul dintre performanţă şi fiabilitate.

2.2.1 Organizarea datelor

Organizarea datelor pentru matricele de discuri poate fi partiţionată în două componente ortogonale: schemele de distribuţie a datelor şi mecanismele de redundanţă. Distribuţia de date defineşte translaţia adreselor logice, vizibile extern, în locaţii de stocare din cadrul subsistemului. Sunt două căi pentru a face acest lucru: prin adresarea independentă a fiecărui disc sau prin divizarea discului. Cea din urmă oferă, adesea, grade îmbunătăţite de încărcare echilibrată, transfer paralel de date, sau acces concurent. Mecanismele de redundanţă specifică tipul, scopul, şi locaţia oricărei informaţii redundante din matrice (de exemplu, replicarea datelor şi paritatea divizată). Matricele redundante de discuri continuă să ofere acces deplin la date şi după defectarea unui disc şi în timp ce datele pierdute sunt reconstruite pe un disc de înlocuire. Multe matrice de discuri combină divizarea discului cu redundanţa pentru a oferi atât performanţă cât şi fiabilitate.

Produsele cu matrice de discuri furnizează, de obicei, *software*-ul şi *firmware*-ul necesare care permit cumpărătorului să aleagă din mai multe scheme posibile de organizare. Deoarece aceasta lărgeşte domeniul de folosire şi îi creşte performanţa relativă la cost, administratorii de sistem trebuie să înţeleagă compromisurile care apar pentru a putea exploata pe deplin posibilităţile matricei.

2.2.1.1 Distribuţia datelor

Distribuţia de date constă în maparea adreselor logice folosite de sistemul gazdă pe discurile din subsistem. Există trei metode principale folosite în realizarea acestei translaţii. Metoda convenţională este de a adresa, independent, fiecare disc şi de a mapa numerele de bloc logic în numere de bloc de disc, direct. Distribuţia este făcută manual de către administratorii de sistem, programe de aplicaţii sau sisteme *software*. Divizarea de disc, numită şi *disk striping* sau *disk interleaving*, reuneşte mai multe spaţii de adrese de disc într-un singur spaţiu, unificat, văzut de sistemul gazdă. Aceasta se face prin distribuirea consecutivă de unităţi logice de date (unităţi de divizare) din cadrul discului într-o manieră *round-robin*. Se disting între două tipuri diferite de divizări de discuri: divizare fină, în care discurile cooperează toate pentru

satisfacerea unei cereri şi divizare grosieră, în care discurile cooperează la cereri mari de date şi lucrează independent la cereri mici de date.

2.2.1.2 Adresarea independentă

Dacă fiecare disc este adresat independent, utilizatorii sau sistemele gazdă trebuie să distribuie explicit datele. În general, administratorii de sistem sunt responsabili pentru răspândirea datelor pe fiecare disc. Încărcarea echilibrată sau distribuirea datelor astfel încât să se încarce echilibrat sistemul, este mai mult o artă decât o ştiinţă şi necesită specialişti. Adrese de date adiacente creează *hot spots* făcând *task*-ul complex.

DISC 0	DISC 1	DISC 2	DISC 3
A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
...	...	...	...

Figura 2-1

2.2.1.2.1 Divizarea fină

Aici, toate cele *n* discuri conţin o fracţiune din fiecare bloc accesibil. Numărul discurilor şi mărimea unităţii de divizare sunt, în general, alese ţinând cont de cea mai mică unitate de date accesibilă. Mărimi mai folosite sunt un bit, un octet, un sector. Mărimea folosită nu este neapărat importantă deoarece fiecare unitate accesibilă este răspândită pe toate discurile. Încărcarea este echilibrată, deoarece toate discurile primesc aceeaşi încărcare. Rata de transfer este de aproximativ *n* ori mai mare faţă de cea a unui disc individual, deoarece fiecare disc transferă $1/n$ din datele cerute. Totuşi, componentele de poziţionare ale capului ale timpului de acces - căutare şi rotire - sunt ori neafectate ori crescute (dacă discurile sunt nesincronizate). De asemenea, doar o singură cerere poate fi făcută la un moment dat, deoarece toate cele *n* discuri lucrează la aceeaşi cerere. Aceste limitări restrâng, de obicei, folosirea metodei la domenii în care timpii de transfer domină timpii în care nu se lucrează cu discul.

DISC 0	DISC 1	DISC 2	DISC 3
A0	A0	A0	A0
A1	A1	A1	A1
A2	A2	A2	A2
...	...	...	...
A7	A7	A7	A7
...	...	...	...
B0	B0	B0	B0
B1	B1	B1	B1
...	...	...	...

Figura 2-2

2.2.1.2.2 Divizarea grosieră

Cu aceasta, nu mai este necesar ca toate discurile să coopereze la fiecare cerere. Această tehnică exploatează paralelismul transferului de date pentru cereri mari, în timp ce permite discurilor separate să satisfacă cereri mici concurent. Câteva efecte se combină pentru a oferi o încărcarea echilibrată şi automată. Primul efect este acela că un fişier va tinde să aibă blocurile sale, componentele împrăştiate pe mai multe discuri. Divizarea grosieră, ca la *hashing* efectiv, randomizează primul disc accesat pentru fiecare cerere, celelalte discuri fiind identificate în maniera *round-robin*. Această încărcare, echilibrată statistic, rămâne intactă şi în cazul adreselor de date adiacente şi distribuţii rapid schimbătoare de accese. Din motivele de mai sus, metoda are posibilitatea să ofere performanţă bună cu un minim de întreţinere.

DISC 0	DISC 1	DISC 2	DISC 3
A0	A1	A2	A3
A4	A5	A6	A7
...	...	...	...
B0	B1	B2	B3
B4	B5	B6	B7
...	...	...	...

Figura 2-3

2.2.1.2.3 Alegerea mărimii unităţii de divizare

Realizarea performanţei de la metoda de mai sus necesită alegerea corectă a mărimii unităţii de divizare. Această alegere determină, pe de o parte, numărul de discuri peste care sunt răspândite datele fiecărei cereri. Mărimi mari ale unităţii permit ca discuri separate să satisfacă cereri multiple (concurent), reducând timpul general de poziţionare, rezultând adesea reducerea cozilor de aşteptare şi o rată de transfer mai mare. Pe de altă parte, unităţile mici determină mai multe discuri să acceseze datele în paralel, reducând timpii de transfer pentru cereri individuale. Acest compromis între paralelismul transferului şi concurenţa accesului este guvernată de mărimea unităţii de divizare.

Când avem cereri de lungime variabilă, fără o secvenţializare sau localizare a adreselor, parametrul cheie este cantitatea de concurenţă din încărcare. Mărimea unităţii de divizare ar trebui să crească o dată cu rata de sosire a cererilor, deci descrescând numărul mediu de discuri accesate pe cerere.

Pentru cereri constând în primul rând din cereri aliniate, de mărime fixă, mărimea unităţii de divizare poate fi aleasă astfel încât numărul de accese la disc, pe cerere, să fie constant. În particular, folosind orice multiplu de mărimea fixă a cererii, se permite fiecărei cereri să fie servită de către un singur disc.

Pentru încărcări cu un oarecare grad de aliniament sau localizare a adresei, echilibrarea trebuie, de asemenea, considerată. Când cererile sunt mari, echilibrarea nu afectează semnificativ alegerea mărimii unităţii de divizare. Pentru încărcări mici, unităţile mari exploatează

beneficiile timpului de poziţionare datorate localizării şi secvenţialităţii adreselor.

Un compromis mai complex este cel pentru încărcări generale, adesea caracterizate prin cereri mici, cereri mari scurte, şi adrese puternic adiacente. În aceste cazuri, abilitatea metodei divizării grosiere de a echilibra încărcarea se îmbunătăţeşte pe măsură ce mărimea unităţii de divizare scade. Mărimi mai mari ale unităţii tind să sufere din cauza formării unor *hot spots* de scurtă durată. S-a sugerat că unitatea de divizare trebuie să fie de 10 ori mai mare decât mărimea medie a cererii, rotunjită la un multiplu al mărimii pistei. Alegerea mărimii unităţii divizare poate fi rafinată considerând informaţie adiţională, cum ar fi distribuţiile de mărime a cererii şi caracteristicile localizării.

O întrebare interesantă este dacă mărimea unităţii de divizare trebuie să fie multiplu de mărime a pistei. Pe de o parte, o cerere pentru unitatea de divizare completă are avantajul unui disc suportând acces cu latenţă zero. Pe de altă parte, mărimea pistei poate corespunde, în mod dificil, mărimii şi aliniamentului cererii, rezultând cereri multidisc, excesive. Multe încărcări constau din cereri de blocuri de mărime specifică, să zicem 4k sau 8k. Mărimile curente de piste sunt de multipli de sectoare (adesea 512 octeţi) alese, mai degrabă, să maximizeze capacitatea decât să potrivească limitările datorate alinierii. Cele mai multe unităţi de disc *state-of-the-art* folosesc înregistrarea pe zone multiple pentru a creşte densitatea de stocare. Folosind aceste discuri, potrivirea mărimii unităţii de divizare cu cea a pistei cere o mărime diferită pentru fiecare zonă de disc. Sunt necesare mai multe studii de performanţă înainte ca divizarea bazată pe piste să poată fi identificată ca o alegere bună, în general.

2.2.2 Mecanismele de redundanţă

Adăugând mecanisme de redundanţă la un sistem de stocare, invariabil îi creşte costul, i se reduce capacitatea, şi/sau i se reduc performanţele. În schimb, multe sisteme fac, periodic, copii ale datelor critice, de obicei, pe banda magnetică. Orice informaţie coruptă sau pierdută din sistemul primar este restocată de pe copie şi orice actualizări care au apărut după efectuarea copiei sunt irecuperabile. Din fericire, căderile individuale de discuri sunt foarte rare, iar căderile de sistem sunt şi mai rare. Timpul mediu dintre defectări (MTBF) la un disc modern este de la 200000 până la 1 milion de ore sau, altfel spus, 20-100 de ani.

Totuşi, pe măsură ce numărul de discuri continuă să crească, MTBF-ul unui sistem tipic neredundant se micşorează până la perioade de luni sau chiar săptămâni. Presupunând că MTBF-ul fiecărui disc este independent şi exponenţial distribuit, MTBF-ul unui grup este invers

proporţional cu numărul discurilor. Aceasta ameninţă fezabilitatea sistemelor nonredundante de discuri.

În plus, numărul de discuri afectate de cădere creşte dramatic o dată cu folosirea divizării discurilor. Un subsistem divizat poate pierde $1/n$ din fiecare fişier, faţă de $1/n$ fişiere din matricea de discuri în cazul adresării independente. Un subsistem cu redundanţă în organizarea datelor poate supravieţui căderii uneia sau mai multor componente. Totuşi, la apariţia unor noi căderi va rezulta inaccesibilitatea sau pierderea datelor. O dată cu creşterea subsistemelor de stocare, redundanţa *on-line* va fi necesară pentru datele cruciale, devenind, probabil, un standard pentru stocare, în general.

2.2.2.1 Replicarea datelor

Cel mai simplu mod de a proteja datele este replicarea lor. Copii separate ale fiecărui bloc sunt menţinute pe discuri separate. Datele devin indisponibile doar dacă toate discurile, conţinând o copie, cad. Această formă de redundanţă, folosită de mai mulţi ani, se numeşte *disk mirorring*, *disk duplexing*, *disk shadowing* sau, mai nou, RAID 1(Figura 2-4). Costul replicării datelor este proporţional cu *d-1*, deoarece costul discului domină preţul subsistemelor de stocare. Deoarece două copii oferă înaltă fiabilitate, alte copii, în plus, nu se folosesc. De fapt, costul mare al replicării obligă, adesea, la folosirea altor mecanisme de redundanţă care nu dau insa acelaşi nivel de performanţă şi fiabilitate.

DISC 0	DISC 1	DISC 2	DISC 3
0	0	2	2
1	1	3	3

Figura 2-4

Menţinerea unor copii multiple ale datelor afectează performanţa în diferite moduri. Primul este acela că scrierile trebuie făcute pe toate copiile. Depinzând de implementare, aceasta poate crea oportunităţi pentru pierderea datelor sau timpi crescuţi de răspuns. Performanţa la citire este, pe de altă parte, mai bună în acest caz. În primul rând, cererile de citire se satisfac simultan, crescând rata de transfer şi scăzând timpii de aşteptare. În plus, o cerere de citire poate fi programată pe discul cu cel mai mic timp de acces. Această performanţă bună la citire poate face raportul cost/performanţă chiar mai mic decât în cazul

sistemelor cu discuri nonredundante, deşi costul absolut este de *d* ori mai mare.

DISC 0	DISC 1	DISC 2	DISC 3
0	1	2	3
3	0	1	2

Figura 2-5

Metoda convenţională de implementare a replicării datelor dublează fiecare disc (Figura 2-4). Fiecare set de *d* discuri identice pot supravieţui la *d-1* căderi fără pierdere de date. Într-o matrice cu seturi multiple replicate, pot apărea multe căderi, fără să cauzeze vreo pierdere de informaţii, atâta timp cât *d* din ele nu sunt din acelaşi set unic, replicat. Totuşi, fiecare cădere de disc reduce performanţele unui set relativ la alte seturi. Acest dezechilibru poate duce la scăderea performanţelor generale ale sistemului, dacă un set căzut devine o gâtuire.

DISC 0	DISC 1	DISC 2	DISC 3
0	1	2	3
1c	0a	0b	0c
2b	2c	1a	1b
3a	3b	3c	2a

Figura 2-6

Alternativ, seturile se pot suprapune astfel încât această degradare de performanţă este împărţită în mod egal de toate discurile rămase. Termenul *declustering* se referă la o metodă pentru combinarea seturilor multiple replicate. Metoda *chained declustering* se referă la divizarea fiecărui disc în *d* secţiuni. Prima secţiune conţine copia primară a unor informaţii, cu fiecare disc având date primare diferite (Figura 2-5). Cu o aranjare inteligentă, această metodă poate menţine o încărcare echilibrată după o cădere a unui disc, deşi probabilitatea supravieţuirii la căderi multiple este redusă.

Al doilea tip de *declustering*, cunoscut şi sub numele de interpusă partiţionează, de asemenea, fiecare disc, dar divizează copiile

nonprimare din toate discurile în loc să le păstreze în spaţii contigue. Aceasta echilibrează încărcarea suplimentară cauzată de o cădere de disc, dar reduce, în continuare, fiabilitatea (pierderea oricăror *d* discuri rezultă într-o cădere critică, (Figura 2-6).

2.2.2.2 Protecţia bazată pe paritate

Proiectanţii de subsisteme de discuri s-au folosit de rezultatele studiilor extinse ale codurilor detectoare şi corectoare de erori. Din teoria codurilor se ştie că pierderea unui singur bit poate fi recuperată ştiind bitul de paritate, ceilalţi biţi de protejat şi faptul că acel bit este eronat. Deoarece sistemele actuale de discuri au o logică de control care, în general, detectează orice cădere de disc, poate fi folosită pentru protejarea unui set de discuri faţă de erorile singulare. Fiecare bit de paritate este calculat din bitul de date asociat de pe fiecare disc protejat şi stocat pe un disc separat. Locaţiile unor biţi particulari de date daţi de un bit de paritate pot varia de la schemă la schemă, aşa cum poate varia şi aranjarea parităţii. Capacitatea adiţională cerută (un disc complet) pentru a menţine redundanţa prin intermediul parităţii este considerabil mai mică decât atunci când se foloseşte replicarea datelor. Deşi costul este rezonabil, menţinerea parităţii poate reduce în mod semnificativ performanţa. Cererile de citire sunt efectuate ca în matricele nonredundante. Dacă se adaugă un disc pentru a oferi capacitate de stocare pentru paritate, el oferă servicii în plus pentru cereri de citire (în cele mai multe scheme de aşezare a parităţii). Cererile de scriere necesită lucru suplimentar cu discul pentru actualizarea parităţii, adesea ducând la o performanţă scăzută. Dacă toţi biţii de date asociaţi cu un bit de paritate sunt actualizaţi de o cerere de scriere, noul bit de paritate este calculat şi inclus ca o parte a cererii totale. Dacă doar un subset al biţilor de date este scris, actualizarea bitului de paritate necesită lucru suplimentar.

Aceasta constă în citirea biţilor de pe discuri pentru construirea noului bit de paritate. Avem două metode:

- **Read-modify-write,** unde noul bit de paritate este construit din valorile vechi şi cele noi ale biţilor care se scriu;
- **Regenerate write,** în care se generează bitul nou de paritate din valorile tuturor biţilor (adică noile valori ale biţilor care se scriu şi valorile curente ale biţilor neschimbaţi).

RMW necesită, mai întâi, accese de citire la toate discurile care se actualizează (incluzând discul care conţine paritatea), pe când R-W cere accese de citire la toate discurile care nu se actualizează. Pierderea

performanţei poate fi redusă prin combinarea dinamică a acestor două opţiuni.

DISC 0	DISC 1	DISC 2	DISC 3
A	A	A	a
B	B	B	b
C	C	C	c
...	...	...	...

Figura 2-7

Datorită nevoii de a face citiri de pe disc înainte de actualizarea parităţii, performanţa poate fi semnificativ mai mică faţă de un sistem neprotejat sau faţă de un sistem protejat prin replicarea datelor. Reducerea cât mai mult a acestei degradări de performanţă este, încă, un subiect aflat în cercetare.

DISC 0	DISC 1	DISC 2	DISC 3
A	A	A	a
B	B	b	B
C	c	C	C
...	...	...	...

Figura 2-8

Există trei scheme majore pentru împrăştierea informaţiei de paritate în cadrul discurilor din subsistem. Metoda directă este de a plasa toţi biţii de paritate pe un disc dedicat de paritate (Figura 2-7). Deşi simplifică maparea adreselor logice în adrese de disc, fiecare scriere trebuie să actualizeze biţii asociaţi de pe unicul disc de paritate. Matricele care folosesc divizarea fină sunt potrivite pentru această metodă, deoarece ele satisfac doar o cerere la un moment dat. Cu alte scheme de distribuţie, totuşi, discurile pot limita performanţa, mai ales când vechea paritate este cerută să participe în operaţiile de la RMW. Pentru evitarea acestei potenţiale gâtuiri, informaţia de paritate poate fi divizată printre discuri (Figura 2-8). Folosind divizarea parităţii, matricea poate efectua mai multe actualizări în paralel.

DISC 0	DISC 1	DISC 2	DISC 3
A	A	A	a
B	b	C	B
B	C	C	C
...	...	...	...

Figura 2-9

O a treia opţiune s-a propus: *declustered parity*. Pentru înţelegerea acestei scheme ne putem imagina combinarea câtorva matrice mai mici, protejate de paritatea divizată. Datele şi paritatea fiecărei matrice sunt distribuite în întregul set de discuri disponibile (Figura 2-9). Cu un algoritm inteligent de distribuţie, impactul performanţei datorat pierderii unui disc poate fi împărţit, în mod egal, de toate discurile rămase. Deşi matricea poate supravieţui doar unei singure căderi, ar trebui să se refacă mai repede. Ca o alternativă, aceasta metodă se poate imagina ca o cale de a realiza fiabilitate mai bună şi performanţă în funcţionarea la defecte prin creşterea cantităţii de date de paritate menţinută.

2.2.2.3 Alte scheme

Menţinerea parităţii protejează o matrice împotriva căderii unui singur disc. O matrice cu două copii ale tuturor datelor poate supravieţui până la o cădere pe perechea de copii. În ambele cazuri, datele pot fi pierdute dacă apare o a doua cădere, înainte ca prima să fie reparată. Oferirea protecţiei necondiţionate împotriva căderilor multiple necesită redundanţă suplimentară. În cele mai promiţătoare metode se folosesc două coduri *Reed-Solomon*, corectoare de erori. Cu asemenea redundanţă, o matrice poate reface toate datele după pierderea accesului la oricare două discuri. Compromisurile referitoare la performanţă şi opţiunile de amplasare a redundanţei se potrivesc pentru acele matrice cu protecţie bazată pe paritate, în afară de faptul că doi biţi sunt mai bine menţinuţi decât unul.

Alte scheme, cum ar fi paritatea multidimensională şi codurile mai agresive, corectoare de erori, pot proteja împotriva unui număr mai mare de căderi. Totuşi, costul şi implicaţiile de performanţă, combinate cu faptul că fiabilitatea sistemelor de stocare creşte tot mai mult, pot obliga folosirea lor.

2.2.3 Performanţa versus fiabilitatea

Cererile de scriere pot duce la întârzieri mari între momentul iniţierii lor şi momentul scrierii efective a noii informaţii pe suportul stabil. Până când sistemul gazdă primeşte mesajul de satisfacere a cererii orice resurse de sistem nu pot fi reutilizate. De asemenea, procesele pot aştepta pentru astfel de confirmări, putând rezulta cicluri inutile de procesor. Acceptând confirmarea cât mai repede posibil, gazda poate utiliza mai bine resursele sale şi are, astfel, mai puţine cicluri inutile de procesor, rezultând o performanţă mai bună. Totuşi, în cazul în care confirmarea cererii este dată înainte ca datele să fie scrise efectiv pe disc, sistemul este vulnerabil la coruperea sau pierderea informaţiilor, datorită căderii tensiunii sau defectării componentelor. Când sistemul renunţă la constrângerile de fiabilitate în scopul câştigării performanţei, trebuie luate măsuri de precauţie sau se acceptă consecinţele.

2.2.3.1 Negocierea fiabilităţii în favoarea performanţei

Un controler de matrice de discuri cu un *cache* (sau *buffer*) poate semnaliza confirmarea gazdei de îndată ce ultimul octet a fost copiat din memoria principală. Astfel de scheme îmbunătăţesc mult performanţa văzută de gazdă la scriere, deşi datele sunt vulnerabile până la scrierea pe disc. Managerul de disc, care este responsabil de determinarea ordinii de servire pentru accesele la disc aflate în aşteptare, are, acum, o problemă. Fiabilitatea cere managerului să expedieze accesele la disc care transferă datele din *cache* pe suportul fizic, în timp ce performanţa obligă managerul să amâne această activitate de fundal.

În matricele care folosesc replicarea informaţiilor, raportarea confirmării după ce doar o fracţiune din copii a fost actualizată poate şi ea să îmbunătăţească performanţa. Datele sunt pierdute doar dacă apar căderi multiple înainte să se efectueze scrierile rămase. Pentru matricele protejate prin paritate, punând paritate în *cache* poate să se îmbunătăţească performanţa prin acumularea noii parităţi pentru scrieri multiple şi efectuarea actualizări de paritate într-o singură scriere.

În schemele de paritate care cer cicluri de actualizare, sistemul este vulnerabil la coruperea datelor dacă setul de scrieri pe disc nu se face simultan. În cazul în care cade tensiunea, la un moment nepotrivit, doar câteva cereri vor fi satisfăcute. Aceasta face ca paritatea să fie incorectă, ducând la coruperi ulterioare dacă sunt lăsate necorectate şi apoi folosită la refacerea datelor. Pe de altă parte, impunerea scrierilor simultane necesită comportare sincronă din partea mai multor discuri. Aceasta poate produce programări neoptime ale cererilor şi un timp de disc inutil

crescut. Când controlerul îşi programează, independent, cererile individuale de scriere, utilizarea discurilor poate fi maximizată.

2.2.3.2 Îmbunătăţirea performanţei şi a fiabilităţii

Cei mai mulţi utilizatori nu sunt deranjaţi de coruperea datelor, exceptând cazurile inacceptabile. Există câteva tehnici pentru îmbunătăţirea performanţei şi a fiabilităţii. Ele se realizează prin reducerea latentelor la scriere în timp ce este garantată siguranţa datelor vechi şi se asigură securitatea datelor noi înainte de semnalizarea confirmării. Aceste scheme pot fi implementate independent de sistemul gazda sau în cadrul matrice de discuri. O soluţie simplă este folosirea memoriei RAM *cache*, non-volatilă, pentru a scrie datele sau paritatea. Aceste memorii menţin informaţiile chiar şi după întreruperea tensiunii de alimentare. Această organizare ar trebui să ofere o fiabilitate echivalentă cu cea a unei scheme cu redundanţă.

Remaparea scrierii, o formă de paginare dublă, a fost propusă pentru multe organizări de date diferite. Această remapare constă în alterarea dinamică a mapării adreselor logice în cele fizice, astfel încât cererea de scriere de date este totdeauna plasată la locaţii noi. Prin menţinerea copiei anterioare a blocului logic până la satisfacerea cererii, metoda îmbunătăţeşte fiabilitatea. Desigur, mapările trebuie menţinute fiabil pentru a preveni pierderea informaţiilor. O metodă este stocarea informaţiei în NVRAM. Altă metodă, mai complexă, dar mai puţin scumpă este să mărim fiecare bloc fizic de date cu adresa logică corespunzătoare şi cu un indicator de timp, care pot fi folosite să reconstruiască mapările.

Alegând locaţii libere lângă poziţia curentă a capului de citire/scriere, remaparea scrierii poate reduce întârzierea produsă de căutare şi rotire. Algoritmul pentru alegerea locaţiei disponibile trebuie să prevină consumarea tuturor locaţiilor libere dintr-o anumită regiune şi lăsarea altor regiuni nefolosite. De asemenea, în timp, datele secvenţial logice vor fi disparate pe spaţiul fizic. Ca rezultat, cereri de citire ulterioare s-ar putea să aibă de suferit din cauza întârzierilor de poziţionare dacă nu se face o oarecare relocare pentru a reface spaţiul fizic.

A treia opţiune - X *logging* - scrie informaţia despre actualizări într-o zonă de log înainte de actualizările propriu-zise. Scrierile log sunt secvenţiale şi au, de obicei, un timp de răspuns mai mic, mai ales când se foloseşte un disc log dedicat. După scrierea intrării de log, controlerul de matrice poate, în siguranţă, să transmită un mesaj de terminare către gazdă şi gazda accesează discul în fundal. În caz de cădere, conţinutul log-ului permite repetarea oricărei actualizări incomplete. Informaţia

scrisă în log poate să conţină o descriere scurtă a actualizării, o copie a noii informaţii sau copii ale amândurora. O formă de verificare este folosită pentru a reduce timpul de refacere şi permite refolosirea spaţiului de log.

2.2.4 Tratarea căderilor

Prin menţinerea redundanţei, o matrice de discuri poate supravieţui căderii unuia sau mai multor discuri. În timp ce matricea continuă să ofere acces la toate datele, performanţa ei este aproape întotdeauna afectată de căderi. În plus, fiabilitatea este redusă, deoarece căderile ulterioare pot fi critice. După cădere, datele pierdute trebuie să fie reconstruite de pe discurile rămase şi scrise pe un disc de înlocuire, proces numit refacere. Până când acest proces se termină, matricea este vulnerabilă. Discurile suplimentare, numite rezerve rapide, sunt adesea incluse în matrice ca să permită refacerii să înceapă imediat.

În cele mai multe scheme, pierderea accesului la disc reduce numărul de cereri concurente de citire care pot fi deservite. Matricele care folosesc replicarea de date primesc cereri ca de obicei, dar mai puţine discuri efectuează citiri. De asemenea, scrierea necesită actualizarea unei copii mai puţin. La matricele protejate prin paritate, o cerere de citire spre un disc defect cere o regenerare a datelor lipsă. Fiecare bit de date pierdut poate fi refăcut din bitul de paritate asociat, ceilalţi biţi contribuind la paritate. Deci, lucrul cu cereri de citire necesită accese la toate discurile care conţin aceşti biţi. Performanţa la scriere este şi ea afectată în matricele protejate de paritate. Datorită faptului că este afectată şi vechea informaţie, *Regenerate-Write* trebuie folosit pentru toate actualizările de date pe discul căzut şi RMW pentru toate actualizările pe discurile rămase.

Cea mai simplă formă de refacere reconstruieşte secvenţial informaţiile indisponibile şi le scrie pe un disc de înlocuire. Într-o matrice replicată, datele pierdute sunt copiate de pe un alt disc conţinând aceleaşi date. Într-una cu protejare bazată pe paritate datele trebuie refăcute din biţii de paritate şi cei de date. Dacă cerinţele de fiabilitate prevăd ca timpul de reparare să fie minim, cererile gazdei pot fi anulate în timp ce refacerea are loc la viteză maximă. Altfel, refacerea şi sistemul gazdă concurează pentru deservire, rezultând un alt compromis între fiabilitate şi performanţă. Acest compromis este, în particular, important în refacerea matricelor cu protecţie bazată pe paritate, unde regenerarea datelor cere accese la mai multe discuri. Alegerea unui algoritm de refacere, mărimea unei cereri individuale de refacere, şi complexitatea

circuisticii de coordonare multidisc trebuie luate în considerație când se ajustează procesul de refacere.

Piața de matrice de discuri se extinde pe măsură ce costul unităților de disc scade şi capacitatea de stocare de uz general are nevoie de o creştere. Ca rezultat, matricele de discuri devin un echipament standard pe o scară tot mai mare de maşini. În curând, cerințele de stocare ale utilizatorului mediu pot fi satisfăcute fără folosirea părților mecanice. Până atunci, matricele de discuri pot oferi niveluri necesare de fiabilitate şi performanță.
Dacă subsistemele de stocare continuă să se dezvolte ca şi până acum, nivelul inteligenței încorporate în diferitele lor componente va continua să crească. Comunicația între aceste componente va creşte mult, rezultând optimizări globale, un control mai distribuit, şi noi algoritmi de distribuție care răspund dinamic la mereu schimbătorul flux de cereri.

2.3 Clasificarea subsistemelor de discuri redundante din punct de vedere a fiabilității

Pentru a marca mai precis domeniul meu de interes, voi prezenta o clasificare, din punct de vedere al fiabilității, pentru sistemele de discuri, propusa de RAB [Mass97].

Pentru a putea realiza o clasificare a subsistemelor redundante de discuri din punct de vedere a fiabilității, este necesară prezentare unor criterii [Mass97]:

1) protecție împotriva pierderii de date sau a accesului la date datorat căderii unor discuri
2) reconstrucția conținutului unui disc defect pe un înlocuitor
3) protecție împotriva pierderii de date în timpul întreruperii scrierii
4) protecție împotriva pierderii de date în timpul căderii magistralelor de I/O
5) protecție împotriva pierderii de date în timpul căderii oricărei componente implicate în transferul de date
6) monitorizare unităților rezistente la căderi precum şi semnalizarea căderilor
7) disc activ pentru *swap*

8) protecţie împotriva pierderii de date în timpul căderii memoriilor *cache*

9) protecţie împotriva pierderii de date în timpul căderii tensiunii externe de alimentare

10) protecţie împotriva pierderii de date în condiţii de temperatură în afara domeniul de valori acceptate

11) semnalizarea căderilor de componente şi a căderilor din subsistemele adiacente;

Tip sistem de discuri	Criterii suportate
Rezistent la căderi - FRDS	1-6
Rezistent la căderi plus – FRDS+	1-11
Tolerant la căderi - FTDS	1-13
Tolerant la căderi plus - FTDS+	1-17
Tolerant la căderi plus plus - FTDS++	1-18
Tolerant la dezastre – DTDS	1-19
Tolerant la dezastre plus – DTDS+	1-21

Figura 2-10

12) protecţie împotriva pierderii de date în timpul căderii canalului de transfer;

13) protecţie împotriva pierderii de date în timpul căderii *contoller*-ului;

14) protecţie împotriva pierderii accesului la date în timpul căderii memoriei *cache*;

15) protecţie împotriva pierderii accesului la date în timpul căderii surselor de alimentare;

16) protecţie împotriva pierderii accesului la date în timpul căderii magistralelor de I/O;

17) protecţie împotriva pierderii accesului la date în timpul căderii surselor externe de alimentare;

18) protecţie împotriva pierderii accesului la date în timpul înlocuirii FRU;

19) disc de rezervă activ;

20) protecţie împotriva pierderii accesului la date în zona defectă;

21) protecţie împotriva pierderii accesului de la mare distanţă la date în zona căzută.

Clasificarea subsistemelor de discuri din punct de vedere al fiabilităţii pe care o voi folosi în continuare este data în Figura 2-10.

2.4 Concluzii

Pornind de la metodele de creştere a fiabilităţii prezentate în capitolul 1, în perspectiva conturării unui nou mod de abordare a proiectării sistemelor de calcul tolerante la defecte, prezentul capitol îşi propune o trecere în revista problematicii legate de:

- coduri detectoare şi corectoare de erori pentru discuri;
- matricele discuri redundante şi independente – RAID.

În această parte, teza cuprinde următoarele contribuţii:

1) sistematizarea metodelor generale de distribuţie a datelor în RAID;

2) sistematizarea mecanismelor de redundanţă utilizate în RAID.

3 O ANALIZĂ A ARHITECTURILOR RAID

3.1 RAID level 0

RAID 0 nu este chiar un membru din familia RAID deoarece nu utilizează nici un fel de redundanţă în scopul creşterii fiabilităţii. Totuşi, există aplicaţii în care performanţa, capacitatea şi costul sunt determinante, fiabilitatea ocupând un loc secundar.

În RAID 0 datele sunt distribuite pe mai multe discuri din matrice, astfel încât pot fi deservite în paralel atât mai multe cereri de dimensiuni mici cât şi cereri de dimensiuni mari.

3.1.1 Distribuţia datelor în RAID 0

Datele sunt împărţite în secţiuni (blocuri) de mărime un multiplu al mărimii unui sector, şi sunt distribuite pe toate discurile. Acest mod de distribuţie aduce marele avantaj ca o cerere de mărime *n* secţiuni, unde *n* este numărul de discuri din RAID, poate fi deservită simultan de toate discurile, obţinându-se o rată de transfer de aproape *n* ori mai mare decât în cazul utilizării unui singur disc (Figura 3-1, unde cu *Si* s-a notat secţiunea *i*).

DISC 0	DISC 1	DISC 2	DISC 3
S0	S1	S2	S3
S4	S5	S6	S7
S8	S9	S10	S11
...	...	...	...

Figura 3-1

3.1.2 Comportarea RAID 0 în aplicaţii care solicită rate ridicate de transfer

Performanţa oricărui nivel de RAID depinde, în principal, de tipul de cereri de acces şi de modelul de distribuţie a datelor. Acest fapt este foarte clar la RAID 0 , unde mecanismele de redundanţă nu există, deci nu vor influenţa analiza noastră.

Mai întâi vom considera că se foloseşte un RAID 0 pentru atingerea unei rate mari de transfer, fapt ce presupune îndeplinirea următoarelor cerinţe:

- Întreaga cale de comunicaţie între memoria principală şi discuri trebuie să poată susţine nivelul cerut al ratei de transfer (magistralele memoriei şi I/O, controlerele I/O etc.)
- Aplicaţia trebuie să lanseze cereri care să ducă la o utilizare eficientă a RAID, ceea ce, în acest caz, înseamnă că cererea tipică se va referi la mari cantităţi de date contigue din punct de vedere logic, care să fie mult mai mari decât mărimea secţiunii utilizate, astfel încât să poate fi valorificată posibilitatea accesului în paralel la discurile din matrice.

3.1.3 Comportarea RAID 0 în aplicaţii ce solicită rate ridicate de cereri de transfer

În aplicaţii orientate pe tranzacţii, devine mult mai interesant timpul de răspuns decât rata de transfer. Pentru o cerere individuală, de volum mic, dominante sunt întârzierile datorate componentelor mecanice, în mişcare ale discurilor.

O matrice RAID 0 poate asigura o rată ridicată de execuţii de cereri I/O, prin distribuirea acestora în paralel şi independent, pe toate discurile.

3.2 RAID level 1

În RAID 1 se foloseşte redundanţă prin simpla duplicare a datelor. Aceasta aduce unele avantaje dar şi dezavantaje.

Principalele avantaje constau în:

- cererea de citire poate fi deservită de către oricare dintre discurile care stochează data cerută, de obicei utilizându-se cel care asigură timpii căutare *(seek)* şi rotaţie cei mai mici;
- recuperarea datelor pierdute prin căderea unui disc, se poate executa simplu, în timp real, prin utilizarea copiilor;
- dezavantajul, legat de performanţă, pentru RAID 1 constă într-o uşoară scădere la scriere, care se va executa ţinând cont de timpii maximi de acces şi rotaţie pentru discurile implicate.

Din punct de vedere economic, RAID 1 este cea mai costisitoare soluţie RAID, necesitând o capacitate reală de stocare dublă faţă de cea utilă.

3.2.1 RAID 1 şi variantele sale arhitecturale

În acest subcapitol vom descrie arhitectura RAID 1, cunoscută şi sub numele de *mirrored array* (matrice de discuri „în oglindă"). Vom considera diverse variante care diferă una de cealaltă după modul în care sunt plasate datele pe discuri. Aceste variante sunt *mirrored declustering*, *chained declustering* şi o variantă nouă, *goup-rotate declustering*, care combină avantajele primelor doua metode.

3.2.1.1 Mirrored declustering

În acest model cele *N* discuri sunt configurate ca *N/2* perechi de discuri „în oglindă", perechea *i* fiind denumită *mirror i*. Două copii ale datelor sunt plasate de-a lungul acestor *N/2* perechi. Fiecare pereche conţine aceleaşi informaţii, nefiind neapărat necesară plasarea în aceeaşi locaţie pe cele două discuri. Pentru uşurinţa discuţiei, presupunem că cele două copii sunt plasate în perechi la un deplasament de *m* cilindri. Când deplasamentul este zero, perechile sunt „oglindite fizic". Blocuri logice contigue ale celor două copii sunt alocate secvenţial pe secţiunile celor *N/2* perechi de discuri. *Figura 3-2* prezintă un exemplu, în care fişierul *f*, compus din şase blocuri, este plasat pe o matrice formată din opt discuri. În figură s-a notat cu F_i prima copie, şi cu f_i a doua copie a celui de-al *i*-lea bloc al fişierului. O cerere de citire poate fi satisfăcută de oricare copie, dar o cerere de scriere necesită ca fiecare copie să fie reactualizată. Unul dintre dezavantajele acestei arhitecturi este acela când defectându-se un disc şi aria de discuri lucrează într-unul dintre modurile *failure* sau *rebuild*, tot traficul iniţial orientat spre discul defectat este redirectat spre „oglinda" lui, situaţie care poate deveni fără ieşire.

Sector	*Mirror* 1		*Mirror* 2		*Mirror* 3		*Mirror* 4	
0	F1		F2		F3	g1	F4	g2
1	F5		F6					
...								
I		f1		f2		f3		f4
I+1		f5		f6	G1		G2	
	Disc 0	Disc1	Disc 2	Disc 3	Disc 4	Disc 5	Disc 6	Disc 7

Figura 3-2

3.2.1.2 Chained declustering

Pentru a depăşi anteriorul dezavantaj pentru recuperarea informaţiilor, Hsiao şi DeWitt au propus arhitectura *chained declustering*, în care sunt păstrate două copii fizice ale datelor, denumite *primary copy* (copie primară) şi *backup copy* (copie de rezervă). După cum se arată în Figura 3-3, dacă un bloc de date primar este alocat pe discul *i*, blocul său de rezervă este alocat pe discul *{(i + 1) mod N}*. Ca şi în cazul *mirrored declustering*, o citire este satisfăcută de oricare dintre copii, în vreme ce actualizările necesită accesul la ambele copii.

Bloc								
0	F1	F2	F3	F4	F5	F6	F7	F8
1	F9	F10	F11	F12	F13	F14	F15	F16
...								
i	f8	f1	f2	f3	f4	f5	f6	f7
i+1	f16	f9	f10	f11	f12	f13	f14	f15
	Disc 0	Disc1	Disc 2	Disc 3	Disc 4	Disc 5	Disc 6	Disc 7

Figura 3-3

3.2.1.3 Group-rotate declustering

Profitând atât de *mirrored declustering*, cât şi de *chained declustering*, se propune o a treia arhitectură, numită *group-rotate declustering*, după cum se arată în Figura 3-4. În această arhitectură, prima copie este stocată în acelaşi mod ca la *mirrored declustering*, dar a doua copie este rotită printre cele *N/2* discuri rămase. În mod clar, *group-rotate declustering* poate efectua două citiri „mari" în acelaşi timp precum anteriorul *mirrored declustering*, dar în timpul unei defecţiuni distribuie povara discului defectat în mod egal pe cele *N/2* discuri. Dezavantajul acestei metode este că, în cazul defectării a doua discuri, este mai probabilă pierderea datelor decât în cazurile *mirrored/chained declustering*.

F1	F2	F3	F4	f1	f5	f9	f13
F5	F6	F7	F8	f2	f6	f10	f14
F9	F10	F11	F12	f3	f7	f11	f15
F13	F14	F15	F16	f4	f8	f12	f16
Copie 1				Copie 2			

Figura 3-4

O altă arhitectură, *interleaved declustering*, a fost propusă în literatura de specialitate. Deoarece această arhitectură nu este propice aplicaţiilor în care cererile I/O necesită actualizarea unei mari cantităţi de date, o vom exclude din discuţia prezentă.

3.2.1.4 Amplasarea copiilor pe discuri

Există câteva variante ale acestei arhitecturi depinzând de modul în care cele două copii ale datelor sunt plasate pe disc. Vom studia două dintre acestea, după cum se arată în Figura 3-5. În mod obişnuit, axa unui disc reprezintă o colecţie de platane. Fiecare conţine la rândul său un anumit număr de piste. Un cilindru este totalitatea pistelor aflate în aceeaşi poziţie pe fiecare platan. Pentru discurile unei staţii de lucru, de obicei, capetele de citire/scriere sunt ataşate unui dispozitiv care poziţionează capetele pe pista potrivită, şi doar unul din capete transferă date la un moment dat.

Cele două abordări amintite sunt descrise după cum urmează. În ambele cazuri blocurile care aparţin aceleiaşi copii a fişierului aflate pe acelaşi disc sunt stocate continuu.

- *Horizontal Layout* (HL)

Jumătate din platanele de pe fiecare axă sunt folosite pentru a stoca prima copie, iar restul pentru a stoca cea de a doua copie. Cele două copii ocupă aceiaşi cilindri.

- *Vertical Layout* (VL)

Cilindrii sunt partiţionaţi în două zone continue şi de dimensiune egală. Cele două copii sunt stocate în partiţii diferite pe cilindrii la aceeaşi poziţie relativă în cadrul partiţiilor.

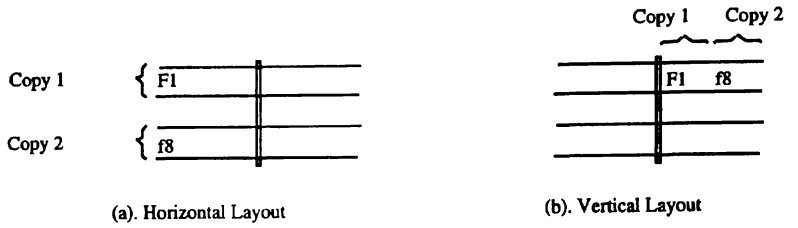

Figura 3-5

De exemplu, considerăm axa unui disc formată din patru platane numerotate de la 0 la 3 şi 1000 de cilindri. Abordarea orizontală plasează copia 1 pe platanele 0 şi 1, ocupând toate pistele acestor două platane, iar copia 2 pe platanele 2 şi 3. Blocurile aparţinând aceluiaşi fişier sunt stocate continuu, exemplu: în cazul Figura 3-4, blocul F9 este adiacent cu F1. Pe deasupra, cele două copii se află şi pe acelaşi cilindru, exemplu: f8 este pe acelaşi cilindru cu F1 (de remarcat că f1, copia identică a lui F1, este pe un disc separat, dar la aceeaşi locaţie ca f8). Pe de altă parte, abordarea verticală plasează copia 1 pe cilindrii 0 până la 499, cuprinzând toate pistele din fiecare cilindru, iar copia 2 pe cilindrii 500 până la 999 (presupunem că fiecare pistă are aceeaşi capacitate). În cazul de mai sus, F1, F9, ..., încep cu cilindrul 0, şi f8, f16, ..., încep cu cilindrul 500 (de remarcat că această reprezentare nu presupune ca spaţiul de pe disc să fie folosit complet, deoarece fiecare grup poate avea cilindri liberi pentru a fi folosiţi ulterior). Abordarea HL favorizează scrierile, în timp ce abordarea VL favorizează citirile la cantităţi mari de date.

Se observă că *mirrored declustering* şi *chained declustering* diferă în felul de tratare a cererilor de citire pentru cantităţi mari de date; *mirrored declustering* poate efectua două astfel de citiri „mari" concurent, în vreme ce *chained declustering* poate efectua doar o singură citire la un moment dat (presupunând că există un singur braţ pe disc). Totuşi, când un disc se defectează, *chained declustering* poate distribui fără probleme „povara discului" defectat către toate cele *N - 1* discuri operaţionale rămase.

3.2.1.5 Un model de disc

Prezentăm modelul de disc care va fi folosit în discuţiile ulterioare. Ne vom concentra asupra timpului de răspuns I/O, care este timpul scurs între sosirea cererii către subsistem şi răspunsul subsistemului.

În mod obişnuit, timpul de răspuns I/O constă în patru componente:

- întârzierea „cozii" la controler *(queuing delaz);*
- timpul de căutare *(seek time);*
- latenta de rotaţie *(rotational latency);*
- timpul de transfer al datelor *(data transfer time).*

Presupunem că fiecare dispozitiv are propria sa cale a datelor. Ne vom concentra doar asupra *queuing delay*, care este în funcţie de încărcarea I/O, şi timpul de acces la disc, care constă în timpul de căutare, de rotaţie şi de transfer.

Se definesc:

- C: numărul de cilindrii pentru fiecare disc;
- N_b: numărul de blocuri de pe fiecare pistă;
- τ: timpul de transfer pentru un singur bloc;
- S_{max}: timpul maxim de căutare;
- R_{max}: timpul unei rotaţii complete;
- D: o variabilă aleatoare *(random variable - r.v.)* care defineşte distanţa căutării pe disc;
- V: r.v.; arată dacă un acces este secvenţial ($V = 0$) sau nu ($V = 1$);
- p_s: probabilitatea ca distanţa de căutare să fie egală cu zero, $p_s=P\{V=0\}$;
- S: r.v.; indică timpul de căutare pe disc;
- R: r.v.; indică latenta de rotaţie;
- X: r.v.; indică suma timpilor de căutare şi de rotaţie, $X=S+R$;
- B: r.v.; indică numărul de blocuri care trebuie transferaţi la o cerere;
- T: timpul de transfer al datelor, $T=\tau B$;
- Y_r, Y_w, Y: r.v.; indică timpii de citire, scriere şi total de pe disc;
- Z_r, Z_w, Z: r.v.; indică timpii de răspuns I/O pentru citire, scriere şi total.

În primul rând să considerăm modelul căutării pe disc. Deoarece s-a observat în realitate că, în majoritatea timpului, braţul discului este nemişcat, vom introduce o probabilitate a accesului secvenţial, p_s, care

se defineşte prin probabilitatea ca distanţa de căutare pe disc să fie egală cu zero. Identificăm două tipuri de accese:

- accese secvenţiale;
- accese nesecvenţiale.

La accese secvenţiale braţul discului nu se mişcă, în vreme ce la accesele nesecvenţiale braţul trebuie să se mişte pentru a răspunde cererii. În acest ultim caz se consideră că braţul se mişcă pe oricare alt cilindru cu probabilitate egală.

Conform cu definiţia lui V, avem în continuare distribuţiile condiţionate pentru distanţa de căutare:

$$P\{D=i|V=0\}=\begin{cases}1 & i=0\\ 0 & i=1,2,\ldots,C-1\end{cases}$$

$$P\{D=i|V=1\}=\begin{cases}0 & i=0\\ \dfrac{2(C-1)}{C(C-1)} & i=1,2,\ldots,C-1\end{cases}$$

unde termenul $2(C - i)/[C(C - 1)]$ rezultă din presupunerea că braţul şi cilindrul ţintă sunt localizaţi aleator pe disc şi că sunt distincţi.

Îndepărtarea câmpurilor de condiţie conduce la:

$$P\{D=i\}=\begin{cases}p_s & i=0\\ (1-p_s)\dfrac{2(C-i)}{C(C-1)} & i=1,2,\ldots,C-1\end{cases}$$

Se foloseşte, de asemenea, şi următoarea expresie pentru timpul de căutare S, ca funcţie de distanţa de căutare D:

$$S=\begin{cases}0 & D=0\\ a+b\sqrt{D} & D>0\end{cases}$$

unde a este timpul de accelerare a braţului şi b este factorul de căutare mecanic.

Pentru uşurinţa analizei, aproximăm că distanţa de căutare D este continuă, cu funcţia de densitate a probabilităţii (*pdf*):

$$f_D(x)=\begin{cases}p_s u_0(x) & x=0\\ (1-p_s)\dfrac{2(C-x)}{C^2} & 0<x\le C\end{cases}$$

unde $u_0(x)$ este funcţia impuls unitar.

Funcţia de distribuţie cumulativă a lui S, $F_S(x) = P\{S < x\}$:

$$F_S(x) = \begin{cases} F_D(0) & 0 \le x \le a \\ F_D\left(\left(\frac{x-a}{b}\right)^2\right) & a \le x \le S_{\max} \\ 1 & S_{\max} \langle x \end{cases}$$

unde timpul de căutare maxim este $S_{max} = a + b\sqrt{C-1}$. Timpul de căutare mediu poate fi:

$$E[S] \approx (1 - p_s)\left[a + b\sqrt{C-1}\frac{8}{15}\right]$$

Timpul de rotaţie se presupune a fi uniform distribuit în intervalul $[0, R_{max}]$ cu pdf:

$$f_R(x) = 1/R_{\max} \quad 0 \le x \le R_{\max}$$

Fie $X = S + R$ suma timpilor de căutare şi de rotaţie, atunci pdf-ul lui X este:

$$f_X(x) = \int_0^{S_{\max}} f_R(x-s)dF_S(s)$$

Pentru parametrii care ne interesează (a se vedea sfârşitul acestei părţi), $R_{max} <= b\sqrt{C-1}$ şi $f_x(x)$ ia următoarea formă:

$$f_X(x) = \begin{cases} f_R(x)p_s + \int_{0+}^{x} f_R(x-s)dF_S(x) & 0 \le x \le R_{\max} \\ \int_{0+}^{x} f_R(x-s)dF_S(s) & R_{\max} < x \le R_{\max} + a \\ \int_{x-R_{\max}}^{x} f_R(x-s)dF_S(s) & R_{\max} + a < x < S_{\max} \\ \int_{x-R_{\max}}^{S_{\max}} f_R(x-s)dF_S(s) & S_{\max} < x \le S_{\max} + R_{\max} \end{cases}$$

Timpul de servire a discului pentru o cerere obişnuită este:

$$Y = X + T$$

cu media:

$$E[Y] = E[S] + E[R] + E[T] = E[X] + E[T]$$

unde T este timpul de transfer, $T = \tau B$.

În cele din urmă, parametrii discului folosiţi în studiul nostru sunt rezumaţi în cele ce urmează: numărul de cilindri C=12000; rata de

transfer = 3MB/sec; timpul rotaţiei complete R_{max} = 16.7ms; dimensiunea blocului = 4096 octeţi; numărul de blocuri per pistă N_b=12; timpul de accelerare a = 3ms; factorul de căutare b=0.5. Astfel, timpul necesar transferului unui singur bloc este τ =1.3ms.

3.2.2 Strategii de accesare pentru RAID 1

Deoarece există două copii ale datelor stocate în matrice, o cerere de citire poate fi satisfăcută de oricare dintre copii, în vreme ce o cerere de scriere trebuie executată asupra ambelor copii. În continuare, ne vom concentra asupra a trei strategii. Pentru toate aceste strategii descrise mai jos, sunt folosite de către sistem două cozi, câte una pentru fiecare copie (Figura 3-6). Fiecare coadă este servită după modelul primul sosit - primul servit. În momentul sosirii, o cerere de scriere generează două sarcini care implică ambele cozi. Diversele strategii diferă între ele prin modul în care se face alegerea discului pentru citirea datelor.

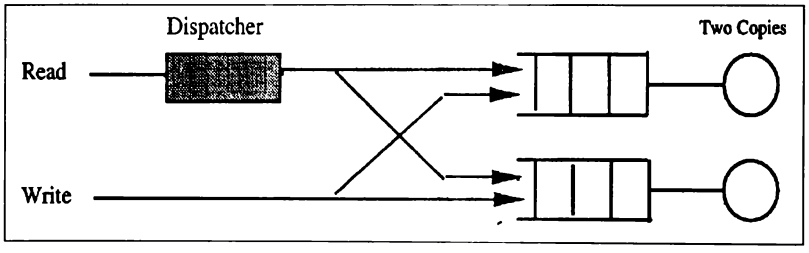

Figura 3-6

3.2.2.1 Strategia Random Join (RJ)

În cadrul acestei strategii, o cerere de citire este îndreptată în momentul sosirii în mod aleator spre coada asociată discului *i* având probabilitatea:

$$\alpha_i,\ i = 1, 2\ (\alpha_1 \quad \alpha_2 = 1).$$

3.2.2.2 Strategia Shortest Queue (SQ)

În cadrul acestei strategii, o cerere de citire selectează discul cu cea mai scurtă coadă în momentul sosirii cererii. Legăturile sunt rupte într-o manieră arbitrară. Când cererea de citire soseşte în cadrul unui sistem gol, ea este îndrumată către discul al cărui braţ este mai apropiat de cilindrul care conţine blocul dorit.

3.2.2.3 Strategia Minimum Seek (MS)

În momentul sosirii unei cereri de citire, strategia MS verifică adresa cilindrului corespunzător ultimei cereri şi îndrumă cererea către coada

cu ceea mai mică distanţă de căutare. Când fiecare coadă este tratată în maniera primul sosit - primul servit, această strategie reduce timpul de căutare pentru cererile de citire.

Remarcă:

> Dintre toate aceste strategii, strategia RJ este cea mai potrivită sistemelor în care cele două discuri conţinând cele două copii ţintă sunt localizate fizic în două poziţii îndepărtate, deoarece nu necesită schimbare de informaţii între cele două discuri. Celelalte strategii, însă, sunt mai potrivite pentru sistemele în care toate discurile din matrice sunt localizate apropiat unele de altele deoarece acestor strategii trebuie să se cunoască starea fiecărei cozi pentru a programa cererile de citire.

3.2.3 Analiza comportării tipurilor arhitecturale RAID 1 cu diverse strategii

În acest subcapitol vom studia şi compara RAID 1 şi variantele sale, *mirrored declustering*, *chained declustering* şi *group-rotate declustering*, în combinaţie cu strategiile RJ, SQ şi MS. În cartea lor, Kim şi Tantawi raportează existenţa câtorva direcţii reale de referinţă a discurilor, care arată că procentajul acceselor secvenţiale se întinde între 25% şi 50%. Pentru matricele de discuri, deoarece datele sunt depuse pe mai multe discuri, probabilitatea de acces secvenţial la fiecare disc este de aşteptat să fie mai mică decât cea a discurilor din sistemele convenţionale. De aici încolo vom alege $p_s=0.2$ în studiul de faţă. Vom considera şi cazul $p_s=0.5$ şi vom observa comportări similare.

Vom începe cu o sumară privire de ansamblu despre cum anume este simulată aria de discuri. În momentul sosirii unei cereri, simulatorul alege o coadă pentru a adăuga datele în concordanţă cu strategia folosită. De exemplu, în cazul folosirii strategiei SQ împreună cu *mirrored declustering*, simulatorul selectează în primul rând aleator un disc dintre $N/2$ discuri, să spunem i, $0<=i<=N/2$, şi selectează perechea discului ca fiind $i+N/2$. Dacă cererea este de citire, ea este îndrumată către cea mai scurtă coadă dintre i sau $i+N/2$. La metoda *group-rotate declustering*, discul pereche este selectat aleator dintre cele $N/2$ discuri. În cazul *chained declustering*, primul disc i este selectat aleator dintre toate cele N discuri, iar discul pereche este $(i+1)\ modN$.

Deoarece cele două blocuri de date duplicate pentru a fi accesate pot fi stocate în diferite locaţii pe discurile pereche, în loc de a păstra informaţii despre locaţia blocului de date, pentru fiecare cerere, simulatorul generează aleator o adresă a unui cilindru pentru un disc

(presupunând că orice bloc este în mod egal accesat) şi apoi obţine adresa cilindrului pereche prin adăugarea unui deplasament. În cazul *chained declustering* cu VL, prima adresă de cilindru este selectată cu probabilitate egală dintre cilindrul 1 până la *C/2*, iar deplasamentul este *C/2*. Pentru alte metode, prima adresă a cilindrului este aleasă cu o probabilitate egală dintre toţi cilindrii. De asemenea, un deplasament de *C/2* este utilizat pentru *mirrored* şi *group-rotate declustering*. Am încercat şi un deplasament egal cu zero, dar diferenţele în performanţe au fost minore.

În cele din urmă, deoarece există o mică diferenţă între performanţele obţinute prin *chained declustering* cu VL şi HL (cu VL evoluând puţin mai bine), vom considera doar performanţele modelului *Vertical Layout*.

3.2.3.1 RAID1 cu strategie RJ

Performanţele sistemelor RAID 1 şi ale variantelor acestuia, în cazul strategiei RJ, sunt prezentate în Figura 3-7_pentru cazurile în care probabilitatea citirii este de $p_r=0.75$, respectiv $p_r=0.25$. După cum se vede în figură, când majoritatea cererilor sunt citiri, toate cele trei strategii prezintă performanţe similare. În cazul în care majoritatea cererilor sunt scrieri, *mirrored declustering* furnizează performanţe uşor crescute în comparaţie cu celelalte două metode, acest lucru deoarece toate datele de pe un disc sunt copiate pe un singur disc. Timpii de căutare corespunzători unei cereri de scriere pe cele două discuri sunt identici atunci când aceasta cerere urmează unei alte cereri de scriere.

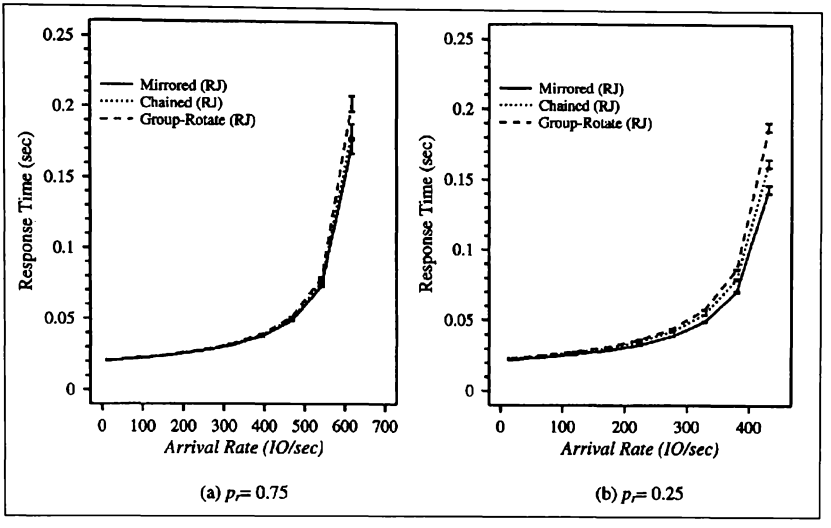

Figura 3-7

În cazul *group-rotate declustering* este, totuşi, improbabil ca timpii de acces la cele două discuri să fie identici deoarece anterioara operaţie de scriere a fost făcută cel mai des doar pe unul din discuri, nu pe ambele discuri. Deoarece operaţia de scriere este considerată încheiată doar după ce se termină operaţiile pe ambele discuri, timpul de răspuns este mai scăzut pentru metoda *group-rotate declustering* decât pentru *mirrored declustering*. Comportamentul în cazul *chained declustering* se află între cele două metode discutate anterior.

3.2.3.2 RAID1 cu strategie SQ

Când se foloseşte strategia SQ, cele trei variante prezintă comportamente foarte diferite (Figura 3-8). Când majoritatea cererilor sunt citiri, se observă că *group-rotate declustering* are rezultatele cele mai bune, iar *mirrored declustering* cele mai slabe. Motivul este că, în cadrul SQ, *mirrored declustering* furnizează o încărcare echilibrată pentru citiri între cele două cozi, iar fiecare pereche oglindită este independentă de celelalte perechi. *Group-rotate declustering* reproduce datele de pe fiecare disc din cadrul a *N/2* discuri. Prin urmare, strategia SQ are abilitatea de a echilibra încărcările pe cele *N/2* cozi.

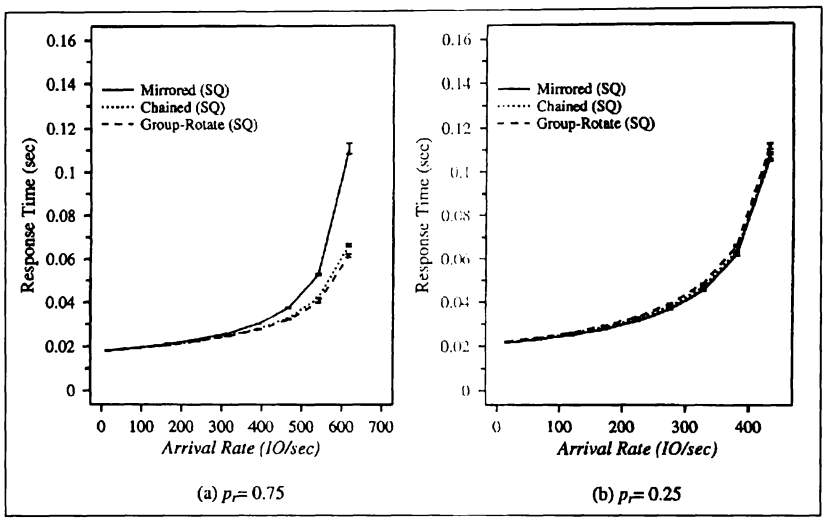

Figura 3-8

Să considerăm cazul *chained declustering*. Deşi datele sunt secţionate pe toate cele *N* discuri, conţinutul fiecărui disc este duplicat doar către cei doi vecini ai săi. Prin urmare, dacă apare un dezechilibru pe unul dintre discuri, problema poate fi uşor rezolvată de către unul dintre vecini.

Deşi cererile de citire beneficiază încă de o încărcare echilibrată când majoritatea cererilor sunt scrieri, timpii de răspuns la cererile de scriere sunt mai mari decât cei în cazul *mirrored* sau *chained declustering*, din motivele enumerate în anteriorul subcapitol. Prin urmare, există doar mici diferenţe între performanţele totale în cadrul celor trei variante.

3.2.3.3 RAID1 cu strategie MS

În cazul strategiei MS, se observă un comportament similar pentru cele trei variante, la fel ca în cazul SQ, (metoda *group-rotate declustering* are performanţele cele mai bune când majoritatea cererilor sunt citiri, iar *mirrored declustering* are performanţele cele mai bune când majoritatea cererilor sunt scrieri.

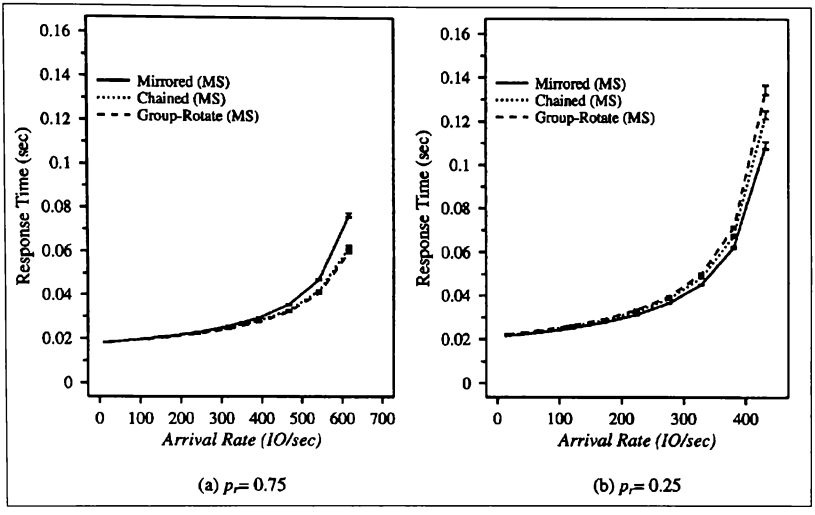

Figura 3-9

3.2.3.4 Analizarea diverselor strategii în cazul Group-Rotate Declustering

Încheiem acest subcapitol prin compararea diverselor strategii în cazul arhitecturii *group-rotate declustering*. Pentru rezultatele prezentate în Figura 3-10 utilizarea discului variază aproximativ de la 0.02 la 0.93. Din figură se observă ca RJ este în mod semnificativ mai neperformant decât celelalte două. Când majoritatea cererilor sunt citiri, sunt doar mici diferenţe între strategiile MS şi SQ. Când majoritatea cererilor sunt scrieri, SQ depăşeşte clar MS. Totuşi, deoarece timpii de servire a discului pentru cererile de citire în cazul MS sunt mai scăzuţi decât în cazul SQ, rezultatul direct va fi mai ridicat în cazul MS.

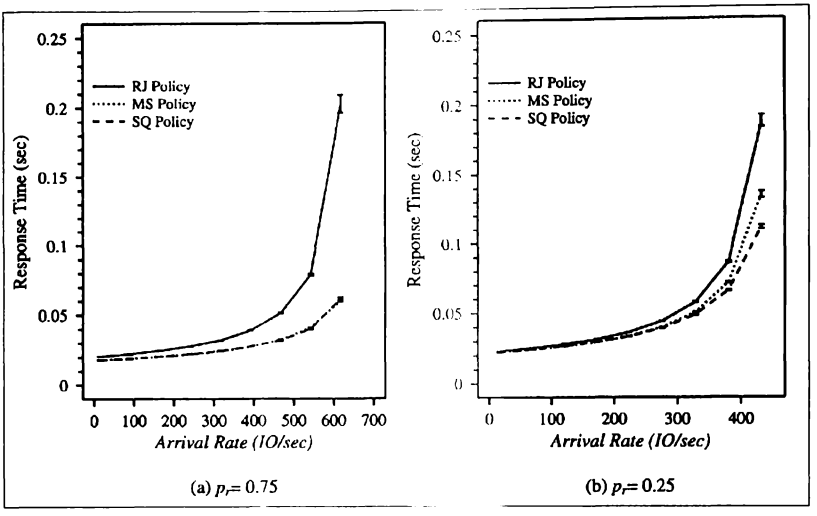

Figura 3-10

3.2.3.5 O analiză a metodei Chained Declustering

După cum s-a descris anterior, există două metode de a plasa datele în cadrul *chained declustering*: *horizontal layout* (**HL**) şi *vertical layout* (**VL**).

HL favorizează operaţiile de scriere deoarece datele din aceeaşi secţiune aparţinând celor două copii sunt localizate în acelaşi cilindru Deci, o scriere masivă poate actualiza cele două copii prin mişcarea braţului către cilindrul destinaţie, actualizând prima copie, urmată apoi de o întârziere de rotaţie, iar apoi actualizând a doua copie (de reamintit că blocurile aparţinând celor două copii aflate pe un disc nu sunt aceleaşi). Prin urmare, este necesară doar o singură căutare. Timpii de servire pentru citiri şi scrieri sunt:

$$Y_r = X + T$$

$$Y_w = X + T + R + T$$

unde T este timpul de transfer şi depinde de numărul blocurilor care trebuie transferate de pe fiecare disc.

Acest sistem poate fi modelat ca o coadă *M/G/1*. Timpul de servire este:

$$Y = \theta Y_r + (1 - \theta) Y_w$$

cu valorile momentane:

$$\overline{Y} = \overline{X} + \overline{T} + p_w(\overline{R} + \overline{T})$$

$$\overline{Y^2} = \overline{X^2} + 2\overline{X}\,\overline{T} + \overline{T^2} + p_w(3\overline{T^2} + \overline{R^2} + 2\overline{X}\,\overline{R} + 2\overline{X}\,\overline{T} + 2\overline{R}\,\overline{T})$$

După formula *Pollaczek-Khinchin*, timpul mediu de aşteptare în coadă este:

$$\overline{Q} = \frac{\lambda \overline{Y^2}}{2(1-\rho)}$$

unde $\rho = \lambda \overline{Y}$ arată modul de folosire a dispozitivului.

În final, timpul mediu de răspuns I/O, $\overline{Z}$, este dat de

$$\overline{Z_r} = \overline{Q} + \overline{Y_r}$$

$$\overline{Z_w} = \overline{Q} + \overline{Y_w}$$

$$\overline{Z} = p_r \overline{Z_r} + p_w \overline{Z_w}$$

unde $\overline{Z_r}$ şi $\overline{Z_w}$ sunt timpii medii de răspuns pentru cereri de citire, respectiv de scriere.

VL favorizează cererile de citire, deoarece cele două copii ale datelor sunt localizate pe cilindri diferiţi, la o distanta de *C/2* piste între copii, unde *C* este numărul total de cilindrii ai fiecărui dispozitiv. O citire poate fi făcută din copia cea mai apropiată de poziţia braţului curent. În orice caz, o citire nu caută niciodată pe mai mult de *C/2* piste. În timp ce strategia VL reduce timpul de servire a citirilor, ea creşte timpul total al operaţiilor de scriere. Când începe o operaţie de scriere, braţul se mută pe copia cea mai apropiată de poziţia sa curentă, actualizează copia şi se mută apoi la *C/2* piste pentru a actualiza şi cealaltă copie. Astfel, timpii de servire a citirilor şi scrierilor $Y_r = X + T$ şi $Y_w = X + T + S_c + R + T$, unde X=S+R este suma timpului de căutare şi a latentei de rotaţie, iar S_c este timpul de căutare a *C/2* piste.

Analog, putem modela sistemul printr-o coadă *M/G/1* şi să calculăm valorile momentane ale timpului de servire total prin

$$\overline{Y} = \overline{X} + \overline{T} + p_w(\overline{R} + \overline{T} + S_c)$$

$$\overline{Y^2} = \overline{X^2} + 2\overline{XT} + \overline{T^2} + p_w(3\overline{T^2} + R^2 + 2\overline{XR} + 2\overline{XT} + 4\overline{RT} + S_c(S_c + 2\overline{X} + 2\overline{R} + 4\overline{T}))$$

Performanţele celor trei variante de RAID 1 sunt prezentate aici făcând uz de modelul anterior prezentat *group-rotate declustering* şi de

simulările pentru modelele *mirrored* şi *chained declustering*. Numărul de discuri din matrice a fost ales $N=64$. Dacă un disc are o capacitate de la 1G la 3G *bytes*, o matrice de discuri cu 64 de astfel de discuri poate atinge o capacitate de la 64G la 192G *bytes*.

Fiecare cerere se presupune că accesează N_s secţiuni plus o secţiune parţială, dimensiunea cerută este B $N_s W+T_s$ blocuri, unde W este lăţimea secţiunii. Presupunem că N_s este o distribuţie geometrică cu parametrul q, iar T_s are următoarea distribuţie:

$$P\{T_s = i\} = \begin{cases} p_f & i = 0 \\ (1-p_f)\frac{1}{W-1} & i = 1,\ldots,W-1 \end{cases}$$

unde p_f este probabilitatea ca o cerere să execute o operaţie I/O pe întreaga secţiune (*full stripe* I/O). Cu alte cuvinte, fiind dată o cerere care conţine şi o secţiune parţială, dimensiunea acestei secţiuni parţiale este uniform distribuită în secţiune. Dimensiunea medie a cererii B este 256 blocuri (1M *bytes*). Atunci valorile momentane pentru N_s sunt:

$$\overline{N_s} = \frac{B - \overline{T_s}}{W}$$

$$N_s^2 = \frac{2 - q}{q^2}$$

unde $\overline{T_s} = (1-p_f)W/2$ şi $q = 1/\overline{N_s}$.

Prin urmare, valorile momentane pentru timpul de transfer pe fiecare disc T, sunt:

$$\overline{T} = (\overline{N_s} + 1 - p_f)\tau$$

$$\overline{T^2} = \left[\overline{N_s^2} + (1-p_f)(2\overline{N_s} + 1)\right]\tau^2$$

Studiind efectul pe care îl are p_f asupra performanţelor se observă că timpul mediu de răspuns este insensibil la modificările lui p_f în cazul RAID 1, acest lucru nefiind valabil şi pentru RAID 5. Rezultatele prezentate în Figura 3-11 sunt obţinute pentru $p_f=0.5$.

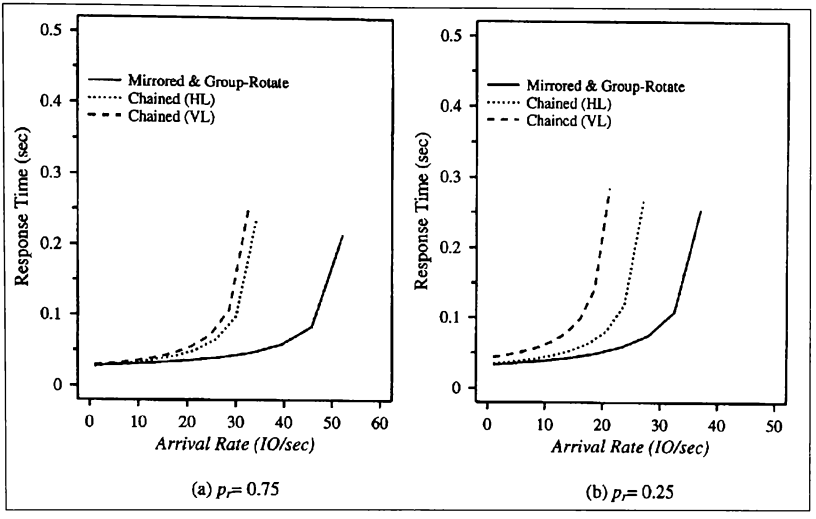

Figura 3-11

Figura 3-11 prezintă timpul mediu de răspuns ca o funcţie a probabilităţilor de citire $p_r=0.75$ şi $p_r=0.25$. În acest caz *mirrored* şi *group-rotate declustering* se comportă la fel şi mai bine decât *chained declustering*. Dintre cele două strategii de plasare a datelor, în cazul *chained declustering*, s-a observat ca HL are rezultate mai bune decât VL din cauza costului ridicat al scrierilor în cazul VL. O dată cu creşterea probabilităţii de citire, performanţele VL se apropie de cele ale HL, şi s-a observat că la $p_r>0.85$, VL depăşeşte performanţele HL.

În Figura 3-12 şi în Figura 3-13 sunt ilustrate procentajele maxime suportate de cele trei arhitecturi, drept funcţie de dimensiunea medie a cererilor şi de numărul de discuri din matrice. În aceste figuri se observă că *mirrored* şi *group-rotate declustering* sunt mai performante decât *chained declustering*.

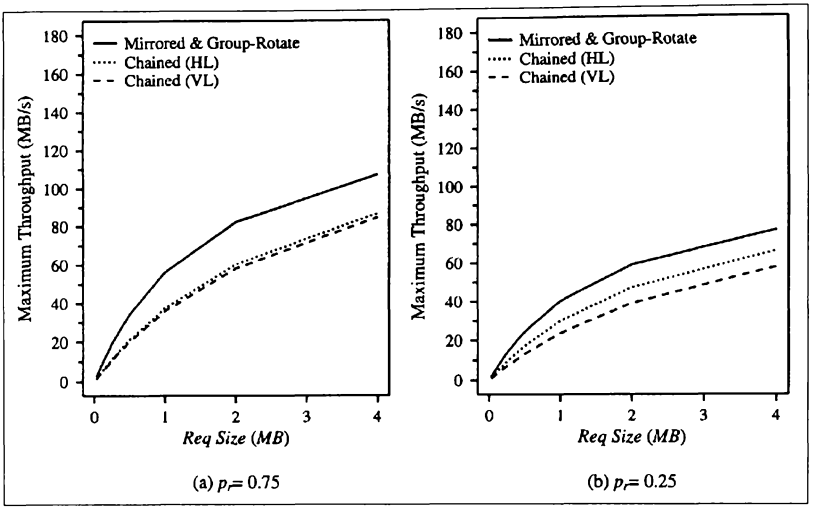

Figura 3-12

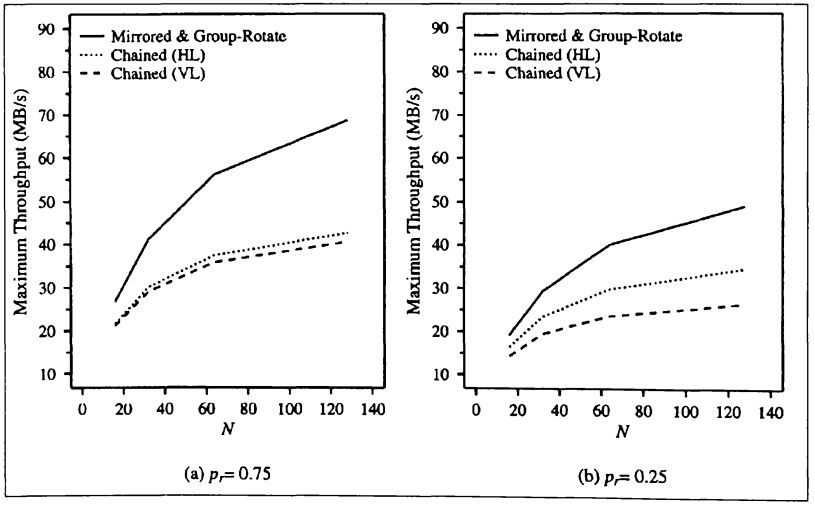

Figura 3-13

Se remarcă faptul că motivul principal pentru care performanţele *mirrored* şi *group-rotate declustering* sunt superioare celor ale metodei *chained declustering* este că, pentru date secţionate de-a lungul celor N discuri, *chained declustering* suportă în mod obişnuit o citire la un moment dat, în vreme ce celelalte suportă două citiri simultan. Însă,

chained declustering poate fi modificată pentru a putea efectua două citiri la un moment dat. Să considerăm o matrice cu 8 discuri. Dacă împărţim discurile 0, 2, 4, şi 6 într-un grup, şi discurile 1, 3, 5, 7 în alt grup, atunci fiecare dintre cele două grupuri conţin blocurile de date necesare pentru o operaţie „mare" I/O (de dimensiune mare). Prin urmare, cele două grupuri pot suporta două citiri de dimensiuni mari în mod concurent. Această abordare creşte, însă, complexitatea *software*-ului şi a sistemelor de fişiere, deoarece fiecare bloc de date citit de pe fiecare disc în parte nu este în ordinea ceruta şi trebuie reordonat în *buffer*-ul memoriei principale. Se pot găsi şi alte abordări pentru a îmbunătăţi performanţele *chained declustering*, dar lasăm explorarea acestora ca un domeniu pentru viitoarele cercetări.

3.3 RAID level 2

RAID 2 utilizează o tehnică paralelă de acces la discurile din matrice, fiecare dintre acestea participând la satisfacerea fiecărei cereri. Datele sunt distribuite pe discurile din matrice în secţiuni foarte mici, de obicei de mărimea unui octet sau cuvânt. Pentru fiecare cuvânt stocat se calculează un cod ECC (uzual *Hamming*), care se depune într-o poziţie omoloagă pe discuri dedicate.

RAID 2 este mai mult o soluţie teoretică decât una practică.

3.4 RAID level 3

RAID 3 este organizată într-un mod similar cu RAID 2, cu deosebirea că se utilizează un singur disc suplimentar pentru memorarea informaţiilor de control (biţi de paritate).

În cazul căderii unui disc, datele de pe acesta se pot reconstrui utilizând datele omoloage existente pe discurile rămase valide. Imediat ce discul defect este înlocuit, datele pierdute se pot reconstrui.

Operaţia de reconstrucţie este simplă. Pentru exemplificare voi considera un RAID cu patru discuri notate D_i, unde pentru $i=3$ considerăm discul cu informaţia de paritate calculată astfel:

$$D_3(k) = D_2(k)\ XOR\ D_1(k)\ XOR\ D_0(k)$$

Presupunând că discul D_1 a căzut, şi adăugând $D_3(k)\ XOR\ D_1(k)$ la ambii membri de mai sus se obţine:

$$D_1(k) = D_2(k)\ XOR\ D_1(k)\ XOR\ D_3(k)$$

În concluzie, datele de pe orice secţiune de pe orice disc, în cazul unei căderi singulare, pot fi refăcute utilizând conţinuturile secţiunilor de pe discurile rămase valide.

În cazul unei căderi de disc, toate datele rămân disponibile, matricea operând în modul redus, cu performanţe diminuate, în special la citire, când se regenerează datele pierdute. Revenirea la performanţa maximă presupune înlocuirea discului căzut şi reconstruirea datelor corespunzătoare pe acesta.

Din punct de vedere al performanţei, datorită faptului că datele sunt distribuite în secţiuni foarte mici, RAID 3 asigură rate foarte bune de transfer. În cazul aplicaţiilor orientate pe rate mari ale cererilor de transfer, performanţa suferă datorită faptului că RAID 3 poate satisface o singură cerere la un moment dat.

3.5 RAID level 4

La RAID 4 se permit accese independente la discuri, astfel încât cereri separate pot fi satisfăcute în paralel. Din acest motiv, RAID 4 este mai potrivită pentru aplicaţii orientate pe număr mare de cereri de acces, decât pentru cele orientate pe transferuri masive.

Şi la RAID 4 se utilizează distribuirea datelor pe discuri diferite, dar secţiunile sunt de dimensiuni relativ mari. Protecţia este asigurată prin paritate, stocată pe un disc dedicat.

Din punct de vedere al performanţei, RAID 4 prezintă o scădere în cazul scrierilor de volum mic, deoarece controlerul de matrice trebuie să actualizeze atât datele, cât şi biţii de paritate. Considerând o matrice de patru discuri, notate cu D_i, unde D_3 este de paritate, rezultă:

$$D_3(k) = D_2(k)\ XOR\ D_1(k)\ XOR\ D_0(k)$$

După actualizarea unei secţiuni, de exemplu pe discul D_1, şi notată ', avem:

$$D'_3(k) = D_2(k)\ XOR\ D'_1(k)\ XOR\ D_0(k) =$$

$$= D_2(k)\ XOR\ D'_1(k)\ XOR\ D_0(k)\ XOR\ D_1(k)\ XOR\ D_1(k) =$$

$$= D_3(k)\ XOR\ D'_1(k)\ XOR\ D_1(k)$$

Rezultă că pentru calcularea noii parităţi, controlerul de matrice trebuie să citească vechea paritate şi veche dată, după care să scrie noua dată şi paritate. Deci, fiecare scriere logică implică două citiri şi două scrieri fizice.

În cazul scrierilor de volum mare, citirile nu mai sunt necesare, noua paritate calculându-se din noile date, astfel că discul de paritate poate fi accesat în paralel cu cele de date şi deci nu mai sunt necesare citiri, respectiv scrieri, suplimentare.

În orice tip de scriere, însă, discul de paritate este implicat, ceea ce uneori poate să genereze întârzieri.

3.6 RAID level 5

În general, RAID 5 este organizată într-un mod asemănător cu RAID 4, cu deosebirea că şi paritatea este distribuită pe toate discurile, eliminându-se, astfel, posibilele gâtuiri întâlnite la RAID 4.

RAID 5, fiind una dintre matricele de discuri cele mai folosite (alături de RAID 1), cercetările mele au fost orientate în special către acestea.

3.6.1 Strategia AR pentru RAID 5

În acest subcapitol, se prezintă o strategie de programare sincronizată a I/O, strategia *after read-out* (AR), pentru RAID 5 care se potriveşte aplicaţiilor în care cererile sunt făcute pentru cantităţi mici de date. Această strategie susţine problema sincronizării scrierilor, descrisă anterior. În toate cazurile, o cerere de scriere se consideră încheiată doar când, atât datele cât şi paritatea, au fost actualizate.

Descriem mai întâi strategia AR pentru cazul când o cerere de scriere necesită actualizarea unui singur bloc. În cazul acestei strategii sunt păstrate două cozi pentru fiecare disc din matrice, una pentru sosirile de cereri de citire/scrieri de date (coada D) şi alta pentru cererile de actualizare a parităţii (coada P), după cum se ilustrează în Figura 3-14. Când soseşte o cerere de citire sau de scriere, dispecerul o trimite în coada D a discului ţintă.

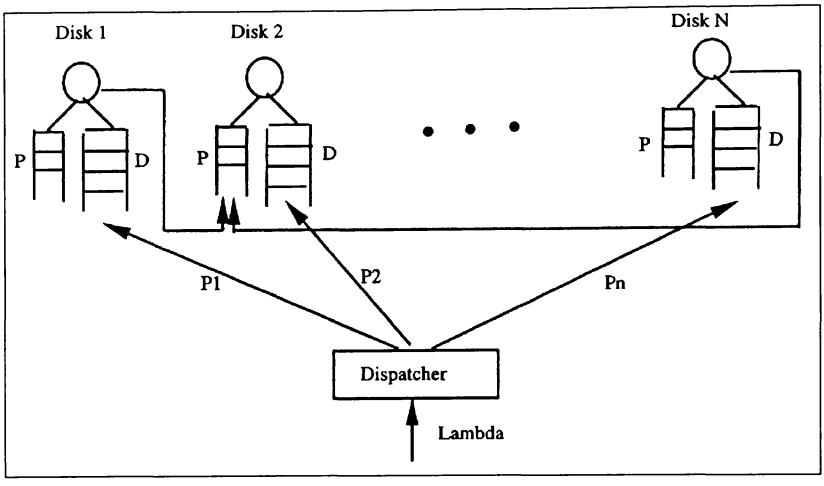

Figura 3-14

Pentru a servi o cerere de scriere, cererile de actualizare a datelor şi a parităţii vor urma următoarele stadii: căutare, rotaţie, citire (pentru calculul noii parităţi), rotaţie completă, şi scris Figura 3-15 prezintă scenariul pentru servirea unei cereri de citire. Când se atinge capul cozii şi este programat pentru servire, se generează o cerere de actualizare a parităţii către coada P a discului corespondent care conţine blocul (blocurile) de paritate, după ce vechile date au fost citite. De aceea, această strategie se numeşte *after read-out* („după citire"). Cererile din coada P au o prioritate mai ridicată decât cele din coada D pentru a fi siguri că o cerere de paritate mai deosebită este servită cel mai curând posibil, deoarece corespunzătoarea operaţie de actualizarea datelor este deja începută. Operaţiile de disc se presupun a fi non-preemptive, deci o cerere de paritate nu poate fi servita până când operaţia I/O curentă nu ia sfârşit.

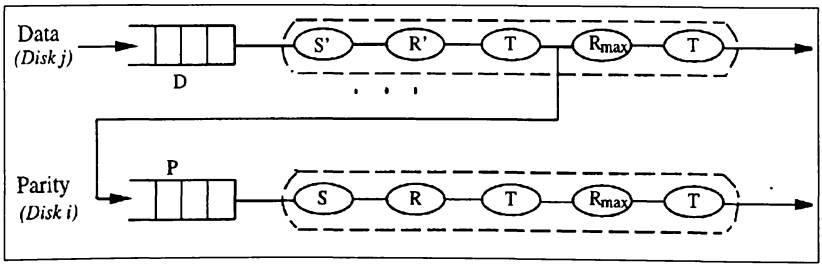

Figura 3-15

Dacă o cerere de scriere doreşte să modifice mai mult decât un bloc dintr-o secţiune, atunci cererea de paritate este generată când ultimul bloc vechi de date a fost citit.

3.6.2 Strategia BS pentru RAID 5

O altă strategie, numită BS, a fost propusă. Aceasta generează o cerere de actualizare a parităţii imediat după ce o scriere îşi începe execuţia. Am observat diferenţe minore între performanţele acestor două strategii (strategia BS se comportă mai bine la încărcări scăzute). Totuşi, strategia AR este mai simplu de implementat, în special în sisteme distribuite în care discurile care conţin blocurile de date şi de paritate pot fi plasate în diferite locuri. Într-un astfel de mediu, strategia AR necesită trimiterea unui singur mesaj unui disc de paritate îndepărtat care conţine datele vechi şi comenzile, în vreme ce strategia BS necesită transmiterea a doua mesaje separate la momente diferite, unul conţinând comenzile, iar celalalt conţinând vechile date.

3.6.3 Tipuri semnificative de RAID 5

3.6.3.1 RAID5/UPGD

Câteva tipuri noi de matrice de discuri au fost recent propuse în care grupurile de paritate au o distribuţie uniformă, astfel încât încărcarea suplimentară creată de căderea unui disc să poată fi împărţită în mod egal de toate discurile supravieţuitoare, rezultând performanţa cea mai bună în modul degradat. În prezent, se utilizează soluţii cu mai multe discuri de rezervă astfel încât să poată fi servite continuu cereri de acces foarte importante. Mai mult, în noile scheme de distribuţie numite distribuţie rezervată, spaţiile de rezervă sunt, de fapt, distribuite în toată aria. Aceasta înseamnă că, după o refacere, noua matrice va fi logic diferită de cea originală.

Arhitecturile RAID 4 şi RAID 5 oferă o toleranţă la defecte prin utilizarea protecţiei prin paritate pentru blocuri de date din diferite discuri. Totuşi, chiar dacă datele unui disc căzut pot fi încă refăcute cu ajutorul parităţii, acesta este un proces încet şi poate duce la creşterea substanţială a încărcării pe discurile supravieţuitoare. Dacă o matrice este proiectată fără luarea în consideraţie a consecinţelor căderii, atunci performanţa sistemului se va degrada la un nivel inacceptabil.

Fie o matrice constând din N discuri în care fiecare grup de paritate implica K discuri. Presupunem că această matrice primeşte o încărcare de R citiri şi W scrieri, distribuite uniform pe matrice. Este evident că în timpul funcţionării normale a matricei încărcarea totală este $R+4*W$, deoarece o scriere implică citirea vechii date şi a vechii parităţi, apoi scrierea noilor date şi a noii parităţi. Când unul dintre discuri a căzut, cererile către acesta sunt deservite de discurile rămase valide în grupurile de paritate corespunzătoare. Se poate arăta că încărcarea totală în modul degradat este $R+4*W+[(K-2)*R+(K-8)*W]/N$, care este suportată de noua matrice, având un disc mai puţin, ca efect al căderii lui. Câteva articole recente au arătat că dacă N este mai mare decât K şi dacă grupurilor de paritate pot fi distribuite uniform în matrice, atunci încărcarea adiţională cauzată de o cădere poate fi împărţită egal de toate discurile rămase. Aceasta este în contradicţie cu ceea ce se întâmpla la RAID 5 standard, cu grupuri de paritate multiple, unde discurile din acelaşi grup de paritate cu discul căzut trebuie să suporte toată încărcarea adiţională, pe când alte discuri nu sunt afectate deloc. Mai mult, se recomandă ca valoarea lui K să fie mult mai mică faţă de N, după cum se vede şi din ecuaţia de mai sus, pentru a rezulta o încărcare generală, ce rezultă în urma căderii unui disc, cât mai mică. Rezultatul net este acela că fiecare disc rămas „vede" doar o mică încărcare suplimentară şi, deci, suferă mai puţin performanţa.

Pasul următor este de a proiecta matrice astfel încât asocierea grupurilor de paritate este distribuită într-o maniera uniformă în matrice (**clustered RAID**). Se defineşte un *BIBD (b,v,r,k,λ) (Balanced Incomplete Block Design)* prin gruparea a v diferite varietăţi de obiecte în b blocuri astfel încât:

- fiecare bloc conţine k varietăţi distincte de obiecte;
- fiecare varietate apare în r blocuri diferite;
- oricare două varietăţi apar împreună în λ blocuri.

Numărând toate replicările de varietăţi, rezultă că $v*r=b*k$.

Similar, considerând numărul de replicări ale varietăţilor dintr-un set de blocuri ce conţine o varietate dată, avem $r*(k-1)=\lambda*(v-1)$.

O matrice generală de paritate cu N discuri este definită după cum urmează. Se partiţionează logic fiecare disc în M domenii de mărime egală. Fiecare grup de paritate implică un domeniu de pe fiecare din cele K discuri, K fiind **dimensiunea grupului de paritate**. Pentru a distribui egal locaţiile de paritate ca în RAID 5, fiecare domeniu este subdivizat

logic în K subdomenii. Un subdomeniu poate conţine mai multe blocuri care nu necesită să fie neapărat contigue fizic ca domeniile, iar subdomeniile sunt considerate entităţi logice (un mod alternativ de a privi lucrurile este că întâi se formează un grup de paritate de bază şi apoi se multiplică în matrice până când toate blocurile sunt atribuite. Toate blocurile dintr-un disc aparţinând aceluiaşi grup de paritate pot fi considerate ca un subdomeniu).

A	A	A	A

Figura 3-16

a1	a1	a1	p1
a2	a2	p2	a2
a3	p3	a3	a3
p4	a4	a4	a4

Figura 3-17

În continuare, vom considera un grup A de paritate, format din patru domenii care va fi notat conform Figura 3-16, înţelegând că subdomeniile sunt organizate conform Figura 3-17 cu locaţiile de paritate rotite. RAID 5 standard este un caz special de matrice generală de paritate, unde $M=1$ domenii pe disc, iar N este un întreg multiplu de K.

O distribuţie uniformă a grupurilor de paritate *(UPGD - Uniform Parity Group Distribution)* este definită ca o matrice generală cu paritate, în care fiecare pereche de discuri are în comun exact L grupuri de paritate. Astfel, dacă oricare disc din aria UPGD cade, toate discurile sunt afectate în mod egal. O matrice de discuri poate fi construită dintr-un BIBD prin mapările:

- varietate->disc
- bloc->grup de paritate,

rezultatul este o matrice UPGD cu $N=v$, numărul de grupuri de paritate $=b$, mărimea grupului de paritate $K=k$, numărul de domenii pe disc $M=r$, şi numărul de grupuri de paritate în comun oricărei perechi de discuri $L=\lambda$. Ca exemplu, vom considera BIBD-ul de mai jos, cu $b=15$, $v=10$, $r=6$, $k=4$ şi $\lambda=2$. Maparea sugerată va duce la o matrice UPGD

de 10 discuri cu 15 grupuri de paritate (de la A la Q), fiecare disc având şase domenii, fiecare grup de paritate având patru domenii şi fiecare pereche de discuri având două grupuri de paritate în comun:

A	B	C	D	E	F	G	H	J	K	L	M	N	P	Q
1	1	1	1	3	2	2	1	2	3	4	1	2	3	4
2	3	4	2	4	4	3	5	6	5	5	8	5	6	7
5	6	7	3	5	6	7	6	7	7	6	9	9	8	8
8	9	10	4	8	9	10	7	7	8	9	10	10	10	9

Figura 3-18

D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
A	A	B	C	A	B	C	A	B	C
B	D	D	D	E	F	G	E	F	G
C	F	E	E	H	H	H	J	K	L
D	G	G	F	K	J	J	M	M	M
H	J	K	L	L	L	K	P	N	N
M	N	P	Q	N	P	Q	Q	Q	P

Figura 3-19

3.6.3.2 RAID 5/VSP

Matricea de discuri oferă avantaje deosebite în privinţa costului, performanţei şi a rentabilităţii. O matrice de *N* discuri mici poate fi mai puţin costisitoare decât un singur disc de capacitate mare, chiar dacă este adăugat un disc pentru informaţia de paritate. Prin adăugarea informaţiei de paritate aria de discuri devine şi mai fiabilă: datele pot fi recuperate la defectarea unui singur disc al oricăruia din matricea de discuri.

Consideraţiile asupra încărcării echilibrate şi asupra uşurinţei recuperării informaţiilor au condus la dezvoltarea unor scheme în care informaţia de paritate este distribuită pe cele *N*+1 discuri. Organizarea clasei de

matrice de discuri în care paritatea este distribuită în acest mod a devenit cunoscută sub numele de RAID 5. În varianta cea mai simplă de organizare a RAID 5 datele sunt organizate în secţiuni cuprinzând *N* blocuri de date şi un bloc de paritate (în care blocul de paritate este, de exemplu, un XOR la nivel de bit a celor *N* blocuri de date). Fiecare bloc din secţiune este plasat pe discuri diferite din matrice, cu poziţia blocurilor de date în secţiune astfel încât fiecare disc să conţină o cantitate egală a blocurilor de paritate.

Deşi această structură oferă câteva avantaje distincte, menţinerea informaţiei de paritate poate introduce o întârziere semnificativă la scriere în cazul unei fragmentări mai mici decât o secţiune. De exemplu, considerând un singur bloc scris într-o secţiune dată: pentru a înnoi informaţia din blocul de paritate sunt necesare informaţii asupra vechiului bloc de date, vechiului bloc de paritate şi noului bloc de date. Dintre acestea toate, cu excepţia noului bloc de date, trebuie citite de pe disc şi noile blocuri de date şi paritate trebuie scrise pe disc, toate acestea reprezentând până la patru operaţii I/O.

Întârzierea celor patru operaţii I/O pentru scrierea unui singur bloc nu trebuie privită ca un dezavantaj.

Dacă se compară timpul celor *N+1* accese la disc per *byte* cu timpul de acces în cazul unui disc de capacitate mai mare. Există cazuri când performanţele pot fi îmbunătăţite reducând acest dezavantaj. Un mod de a realiza acest lucru este acela de a înseria scrierile: dacă toate cele *N* blocuri de date dintr-o secţiune sunt scrise în acelaşi timp, blocul de paritate poate fi calculat în paralel necesitând *N+1* scrieri pe disc (şi nici o citire de pe disc).

O metodă a organizării datelor astfel încât să se beneficieze de proprietatea descrisă anterior este un sistem de fişiere *log-structured* (SFLS).

Oricum, această metodă prezintă un dezavantaj numit „curăţirea segmentului", care necesită periodic citiri şi scrieri de zone relativ mari de pe discuri pentru a se întrepătrunde spaţiul liber pentru următoarea utilizare.

Amintim aici două abordări ale acestei probleme: *floating parity* şi *parity logging*:

- *floating parity* nu reduce numărul total de operaţii I/O; uneori unele perechi de operaţii pot fi realizate într-o singură rotaţie a discului;

- *parity logging* măreşte, de fapt, numărul total de operaţii de I/O, reducând, însă, raportul cost/operaţie ca rezultat al inserării informaţiilor de paritate, dar aceste operaţii necesită un *buffer* de memorie nonvolatil şi un spaţiu adiţional pe disc.

În continuare vom considera o abordare diferită în scopul reducerii costului operaţiei de scriere bazată pe noţiunea VSP a blocurilor de paritate.

Considerând dată o secţiune cuprinzând N blocuri de date $D1, D2, \ldots, DN$ şi un bloc de paritate P, P conţinând paritatea pentru un subset arbitrar de blocuri de date $Di_1, Di_2, \ldots, Di_q$, unde subsetul protejat este identificat în P (plasat, de exemplu, ca un vector de biţi). Informaţia de paritate nu este menţinută pentru blocurile neprotejate de date din secţiune şi astfel de blocuri pot fi scrise şi adăugate setului protejat fără a avea cunoştinţe asupra conţinutului blocului. În plus, dacă blocurile de paritate excedentare sunt plasate într-un *cache* pentru secţiuni conţinând blocuri neprotejate, scrierea sau modificarea unui bloc, dintr-una dintre aceste secţiuni necesită doar două operaţii I/O. În final, recuperarea datelor în urma defectării unui disc este, de asemenea, facilitată deoarece recuperarea blocurilor neprotejate nu este necesară.

Presupunem că doar blocurile de date libere (nealocate) pot fi neprotejate. Acestea pot atinge o stare în care devin protejate şi este de dorit a se trece aceste blocuri în stare neprotejată. Acest proces îl vom numi *cleaning* (curăţire). Acest proces se poate încadra uşor în majoritatea sistemelor de stocare existente implementate prin sisteme de fişiere , sisteme de gestiune a bazelor de date etc.

Pentru a reduce fragmentarea informaţiei stocate, recuperarea blocurilor pierdute, îmbunătăţirea performanţelor discurilor s.a.m.d., aceste sisteme efectuează în mod obişnuit o reorganizare periodică a discului.

Scopul nostru este de a investiga potenţialul îmbunătăţirii performanţei în sisteme utilizând *variable scope protection* (VSP). VSP prezintă grade de libertate în plus, cu menţiunea că se păstrează blocurile de paritate pentru a creşte fiabilitatea. Considerăm două măsuri de evaluare a performanţei: media costului total pentru scrierea unui singur bloc şi costul pentru refacerea informaţiei. Aceste informaţii preliminare se folosesc pentru a obţine mai rapid rezultatele dorite (care se obţin prin simulare sau pe cale analitică). Deşi studiul teoretic arată la prima vedere rezultate nesatisfăcătoare, sistemele care utilizează VSP dau rezultate bune pe cale practică.

3.6.3.2.1 Organizarea unui RAID/VSP

Presupunem un sistem format dintr-o matrice de *N+1* discuri. Acesta conţine un controler cu un *cache*, format dintr-un număr de blocuri (de date şi/sau de paritate) recent accesate. Controlerul gestionează *cache*-ul şi aria de discuri şi procesează cererile provenite de la unul sau de la mai multe sisteme. În general, o colecţie de blocuri de date şi blocuri de paritate asociate este denumită „grup de paritate". O varietate largă de metode propun organizarea grupurilor de paritate în matrice de discuri.

Aici se foloseşte metoda în care fiecare grup de paritate constă dintr-o secţiune, de exemplu un bloc ales din fiecare disc: *N* blocuri de date şi un bloc de paritate. Aceasta nu presupune o reducere a generalităţii problemei (în prezentarea de faţă operaţiile controlerului se aplică la fel de bine pentru orice grup de paritate), dar simplifică prezentarea noastră. Controlerul de disc gestionează secţiunile (sau, mai general, grupurile de paritate): fiind dată o cerere de citire a unui bloc oarecare specificat printr-un identificator de bloc de disc, controlerul determină secţiunea de care aparţine blocul şi discul pe care se află blocul considerat în acea secţiune (desigur, se verifică în primul rând existenţa blocului în *cache*). După cum s-a menţionat în introducere, este de dorit ca, în secţiunile discurilor, blocurile de paritate să fie distribuite în mod egal în cele *N+1* discuri.

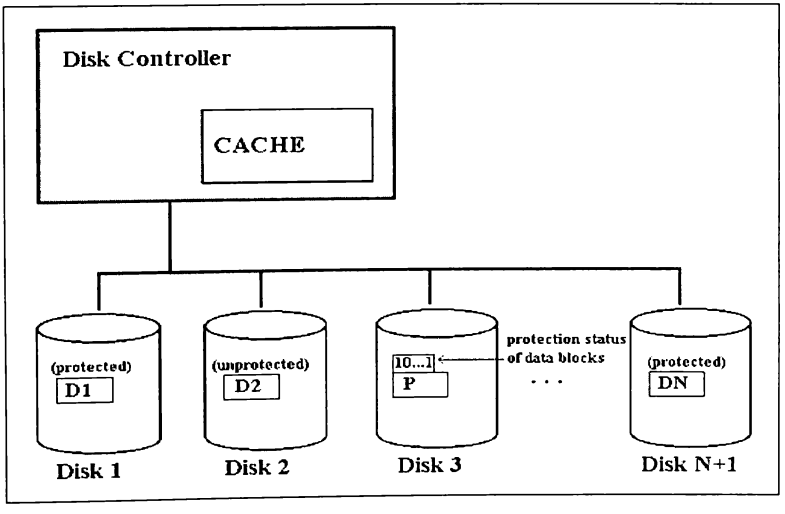

Figura 3-20

Fiecare bloc de date din subsistemul matricei de discuri se poate afla în una din următoarele două stări :

- alocat în acest moment de un sistem de un nivel superior (exemplu: un sistem de fişiere sau un sistem de baze de date) de stocare a datelor;
- nealocat, deoarece fie nu a fost alocat încă, fie a fost anterior alocat, dar la următoarea alocare datele au fost şterse sau mutate în altă parte.

Ne referim la blocurile de date alocate *in-use* (în folosinţă) şi blocurile nealocate *free* (libere). Spre deosebire de arhitecturile RAID anterioare, nu sunt necesare blocuri de paritate pentru toate blocurile de date din secţiune. De asemenea, se presupune că este necesar ca blocul de paritate să conţină informaţii de paritate numai pentru blocurile aflate *in-use*. Poate, de asemenea, să conţină informaţii de paritate pentru un subset oarecare (incluzând şi un set vid) de blocuri goale într-o secţiune. Dacă blocul de paritate conţine informaţii de paritate pentru un bloc de date dat, spunem că blocul de date este protejat; altfel este neprotejat.

În cele ce urmează, fiind dată o secţiune, fiecare bloc de date se poate afla în una din următoarele trei stări:

- *in-use* (obligatoriu protejat);
- *free* şi protejat;
- *free* şi neprotejat.

Dacă toate blocurile de date din secţiune sunt *free*, vom numi aceasta o secţiune *free*. Un subset de blocuri de date aflat într-o secţiune *free* poate fi protejat. Dacă toate blocurile de date dintr-o secţiune sunt *free* şi neprotejate, vom numi aceasta *clean free*. De remarcat că un bloc de paritate a unei secţiuni *clean free* nu conţine nici o informaţie utilă (el conţine informaţii de paritate pentru un set de blocuri vide). Putem considera şi că blocurile de paritate se pot afla în doua stări: *in-use* (unul sau mai multe blocuri de date sunt protejate) sau *free* (toate blocurile de date sunt neprotejate).

Există mai multe moduri de a identifica blocurile de date într-o secţiune. De exemplu: starea fiecărui bloc de date (protejat sau neprotejat) poate fi reţinută într-un vector de biţi aflat în blocul de paritate. Acesta prezintă avantajul pe care îl au blocurile de paritate aflate în *cache* conţinând stări protejate/neprotejate asociate blocurilor de date. Ca un

exemplu pentru un model alternativ, controlerul de disc poate păstra informaţii pentru toate secţiunile într-un RAM nonvolatil.

Este necesară, de asemenea, păstrarea informaţiilor despre stările *in-use* sau *free* ale fiecărui bloc de date. Acesta se asigură la nivelul unui sistem *software* de un strat din sistemul de fişiere sau din sistemul de baze de date, de exemplu. Pentru a fi mai clari, ne vom referi la acesta drept *storage manager*. *Storage manager* implementează următoarele funcţii:

- harta identificatorilor datelor din blocurile de disc (unde aceste blocuri de disc sunt, în contextul de faţă, la rândul lor specificate de identificatorii blocurilor de disc);
- procese care cer citirea sau scrierea datelor;
- procese care cer alocare şi dealocare de date.

De asemenea, controlerul are disponibilă harta identificatorilor blocurilor de disc dintr-o secţiune şi poate, de exemplu, să determine când o intrare dintr-o secţiune este *free*.

În condiţiile actuale este de presupus că *storage manager*-ul este implementat ca un strat în sistemul analizat. De asemenea, se poate implementa *storage manager*-ul ca o parte a controlerului de disc. În acest caz, câteva optimizări devin fezabile. Exemple posibile ar fi:

- integrarea hărţilor de identificatori logici de date ca şi ID-uri a blocurilor de disc şi ID-urile blocurilor de disc ca adrese fizice ale blocurilor.
- integrarea controlului informaţiilor *in-use/free* şi protejate/neprotejate.

Discutarea problemelor atinse anterior nu constituie obiectul lucrării de faţă şi pentru următoarele descrieri presupunem că *storage manager*-ul este implementat independent de controlerul de disc.

Operaţiile obişnuite ale controlerului de disc (în absenţa defecţiunilor) diferă de un controler RAID fără VSP doar în privinţa scrierilor şi a convertirii blocurilor protejate în blocuri neprotejate.

În primul rând vom discuta despre scrieri. Pentru simplitate, vom descrie doar scrierea unui singur bloc (scrierea multibloc este considerată o extensie a scrierii unui singur bloc). Presupunem că paritatea este calculată printr-un XOR pe biţi a blocurilor de date. O scriere într-o secţiune *free* poate fi tratată ca un caz special:

privitor la actuala stare de protecţie a blocurilor de date *free*, după scriere doar un singur bloc *in-use* ar trebui sa fie protejat.

Pentru aceasta, scrierea unui bloc D într-o secţiune liberă S este procesată după cum urmează (unde P este un bloc de paritate).

- se setează $P=D$;
- se marchează D ca protejat în S;
- se scrie P, D pe disc (de remarcat că scrierile pe disc pot avea loc în mod concurent);
- dacă oricare din blocurile de date din S, altul decât D, este prezent în *cache*, se eliberează spaţiul din *cache* pentru fiecare astfel de bloc.

În continuare, scrierea unui bloc D într-o secţiune S *(non free)* cu blocul de paritate P este tratată după cum urmează (unde D_{old} se referă la anterioara versiune a lui D).

Dacă D_{old} nu este protejat, atunci:

- citeşte P de pe disc, dacă nu este în *cache*;
- setează $P=P$ *XOR* D;
- marchează D ca protejat în S;
- scrie P, D pe disc (de remarcat că scrierile pe disc pot fi concurente);
- iesire.

Dacă D_{old} este protejat, se execută următoarele:

- Dacă P şi/sau D_{old} nu sunt în *cache*, atunci citeşte de pe disc (de remarcat că dacă nici unul nu se află în *cache*, citirile de pe disc pot fi concurente);
- Setează $P=P$ *XOR* D_{old} *XOR* D;
- scrie P şi D pe disc (scrierile pot fi concurente);
- Dacă D_{old} se află în *cache*, eliberează spaţiul ocupat de D_{old}.

În cele din urmă se descriu operaţiile pentru „curăţirea“ secţiunilor, prin care se înţelege convertirea blocurilor de date protejate *free* către o stare neprotejată. Avantajul utilizării unei secţiuni curate (în care toate blocurile *free* sunt neprotejate) este la scrierea inseriata în secţiune, care este mai puţin costisitoare în termeni de accese la disc. În cazul unei

scrieri a unei secţiuni *free*, curăţirea unei secţiuni în care toate blocurile de date sunt *free* poate fi tratată ca un caz special, după cum urmează:

- marchează toate blocurile de date din *S* ca neprotejate
- pentru fiecare dintre aceste blocuri de date, dacă se află în *cache* eliberează spaţiul ocupat din *cache*;
- scrie *P* pe disc (presupunând că informaţia de protecţie este depusă în *P*);

Dacă secţiunea de curăţat are blocuri *in-use*, atunci operaţia este puţin mai complicată. Următoarea implementare minimizează numărul de accese la disc în acest caz (unde *S* este secţiunea, *P* este bloc de paritate, blocurile de date *in-use* sunt D_i şi blocurile de date protejate *free* sunt B_j).

Fie *d* numărul celor D_i care nu se află în *cache* şi fie *b* numărul B_j care nu se află în *cache*:

- dacă $d<=b$ atunci citeşte fiecare D_i citeşte fiecare D_i care nu se află în *cache* (citirile pot fi concurente) şi calculează *P* prin *XOR* pe toate D_i;
- altfel, citeşte fiecare *Bj* care nu se află în *cache* (citirile pot fi concurente) şi recalculează *P* prin XOR între *P* şi toţi B_j;

  marchează fiecare B_j ca neprotejat;

  pentru fiecare B_j din *cache* eliberează spaţiul din *cache*-ul blocului;

  scrie *P* pe disc;

Aceasta încheie descrierea operaţiilor controlerului de disc, fără nici un detaliu privitor la algoritmi sau operaţiile la nivelul *storage manager*.

Din cele anterior arătate, se poate observa că pentru anumite operaţii sunt îmbunătăţiri semnificative ale unui sistem fără VSP. De exemplu, o scriere a unei secţiuni *free* îi este necesar doar două accese la disc fără a sonda conţinutul *cache*-ului şi o scriere a unui bloc neprotejat *free* îi sunt necesare trei accese la disc. În mod ideal, *storage manager*-ul ar trebui proiectat în aşa mod încât să beneficieze de avantajele acestor proprietăţi.

Există un număr de idei în modelul unui astfel de *storage manager*, unele dintre ele fiind prezentate în continuare.

Pentru a reduce costurile de scriere, *storage manager*-ul poate să realoce scrierile blocurilor neprotejate *free*. De asemenea, doar dacă secţiunea

este *free*, costul scrierii poate fi redus şi mai mult dacă blocul de paritate se află în *cache*. În consecinţă, pentru dispozitivul de stocare este indicat să se menţină în *cache* un număr de blocuri de paritate pentru secţiunile cu blocuri neprotejate *free*.

După scrierea blocurilor neprotejate *free*, acestea devin blocuri protejate, iar dacă scrierile se datorează realocării, fără alte operaţii, locaţiile anterioare ale blocurilor vor rămâne protejate. Astfel, curăţirea periodică este necesară pentru a converti blocuri protejate în blocuri neprotejate.

Similar, dacă numărul blocurilor de paritate aflate în *cache* pentru secţiuni cu blocuri neprotejate este epuizat, reverificarea acestora este necesară pentru îmbunătăţirea performanţei.

În următoarele două secţiuni, se vor prezenta rezultatele obţinute în legătură cu câteva dintre problemele principale din modelul *storage manager* cu VSP. În primul rând, considerăm un model minimal al *storage manager*, proiectat spre a reduce costul scrierilor prin curăţirea periodică a celor mai puţin utilizate secţiuni, împreună cu scrierea de blocuri neprotejate în aceste secţiuni. Apoi, vom compara această clasă de metode cu cea a sistemului de fişiere structurat (*LFS*) în care scrierile sunt plasate într-un *buffer (bufferate)* şi relocate astfel încât secţiuni întregi (inclusiv blocul de paritate) sunt scrise în acelaşi moment (de remarcat că reorganizarea periodică a discului, operată de *LFS* pentru a întrepătrunde spaţiul liber, este denumită tot „curăţire“. Aici, pentru a evita ambiguitatea, vom folosi termenul „curăţire“ când ne vom referi la conversia blocurilor protejate *free* în blocuri neprotejate, iar „reorganizarea *LFS*“ pentru procesul de întrepătrundere a spaţiului liber de pe disc).

3.6.3.2.2 Analiza performanţei în cazul scrierii aleatoare ale unui singur bloc

Vom descrie performanţele obţinute în simularea unui model simplificat de sistem care foloseşte VSP. Fiecare operaţie se presupune că este o scriere într-un bloc logic ales la întâmplare (uniform distribuit). *Storage manager*-ul conţine o hartă a identificatorilor blocurilor logice corespunzătoare identificatorilor blocurilor de pe disc (identificatorii blocurilor de pe disc sunt transformaţi în adrese fizice de către controlerul de disc). Folosind această hartă, *storage manager*-ul este liber să realoce blocuri, astfel încât fiecare scriere logică poate elibera un bloc *in-use* (precedenta locaţie a blocului logic) şi aloca un bloc *free* (noua locaţie a blocului). Un model mai apropiat de realitate ar putea

include operaţii constând în citiri multiple şi/sau scrieri în aceeaşi secţiune, în care costul modificării blocului de paritate poate fi amortizat prin mai mult decât o scriere.

Presupunem că:

- există *i* blocuri de date *in-use* în orice moment, fiecare specificat prin identificatorul blocului logic 1,2, ...,i care indică spre unul din numărul *d* de blocuri de date din aria de discuri (astfel că raportul de utilizare este de *i/d*);
- fiecare operaţie ulterioară este o scriere a unui singur bloc într-unul din cele *i* blocuri logice, orice bloc fiind la fel de prioritar.

Storage manager-ul execută această secvenţă a scrierii unui singur bloc după cum urmează:

- găseşte cea mai puţin utilizată secţiune, adică secţiunea cu numărul cel mai mare de blocuri *free* (alege orice astfel de secţiune în cazul că sunt mai multe);
- goleşte secţiunea (în modul descris în secţiunea anterioară), convertind toate blocurile protejate *free* în blocuri neprotejate;
- realocă fiecare bloc scris în continuare într-un bloc *free*, până când toate blocurile sunt *in-use*;
- repetă de la pasul 1.

Deoarece blocul de paritate a fost găsit în pasul 2 şi toate scrierile din pasul 3 se fac în blocuri neprotejate, fiecare operaţie de scriere nu necesită nici o operaţie de citire, ci doar două operaţii de scrieri pe disc (una pentru date şi una pentru paritate) care pot avea loc concurent.

Această tactică a fost simulată cu scopul de a găsi costul mediu al curăţirii blocurilor (măsurat în accese la disc) pentru scrierea unui singur bloc. S-au folosit arhitecturile matricelor de discuri 3+1 şi 7+1 (unde *N+1* înseamnă *N* blocuri de date şi un bloc de paritate pentru o secţiune). Pentru a determina acest cost, s-a presupus folosirea metodelor de curăţire prezentate anterior, cu presupunerea cazului cel mai negativ al nici unui *cache-hit* al blocurilor de date. Se deduce că pentru o secţiune *free* costul este nul (un bloc de paritate nou este scris la prima scriere în secţiune) şi pentru o secţiune cu *q* blocuri *in-use* şi *N-q* blocuri protejate *free* costul este mai mic decât *q* sau *N-q+1* citiri de disc (în primul caz sunt citite blocurile *in-use* pentru a calcula paritatea, iar în al doilea caz blocurile neprotejate *free* împreună cu actualul bloc de paritate sunt citite pentru a recalcula paritatea). Nu luăm în calculul

costului scrierea pe disc a blocului de paritate deoarece se presupune că *storage manager*-ul face ca scrierea blocului de paritate să fie întârziată până la scrierea primului bloc de date într-o secţiune (astfel că scrierea blocului de paritate este parte a costului scrierii unui singur bloc).

Simulările au fost făcute pentru sisteme conţinând 1000 şi 10000 de secţiuni. Interesant, rezultatele au fost aproape identice în fiecare caz (totuşi, au fost obţinute rezultate diferite pentru arhitecturile 3+1 şi 7+1). Concluzionăm că, pentru o arhitectură cu un număr dat de discuri, costul mediu al curăţirii la scriere nu depinde de numărul total de secţiuni (presupunând că acest număr este suficient de ridicat). Figura 3-21 rezumă aceste rezultate în care fiecare număr este medie a trei rânduri a câte 25000 de scrieri (a fost observată o foarte mică variaţie între ture).

De remarcat este faptul că la o încărcare de 95% şi peste, costul mediu este acelaşi pentru ambele arhitecturi: 3+1, 7+1. Aceasta se datorează următoarelor: pentru încărcări apropiate de 100%, majoritatea secţiunilor selectate pentru curăţire vor avea un bloc *free*, rezultând în costul de curăţire două citiri de disc (una pentru blocuri protejate *free* şi una pentru blocul de paritate) pentru scrierea ulterioară a unui bloc de date. Astfel, costul mediu devine independent de numărul de discuri la o încărcare aproape de limite şi, pe cât se apropie de o încărcare de 100%, acest cost are o valoare limitată la doi.

	3 + 1 **RAID**		7 + 1 **RAID**	
Stor.	1,000	10K	1,000	10K
Util	Setiune	Setiune	Setiune	Setiune
50%	0	0	0.13	0.13
60%	0.14	0.14	0.30	0.30
70%	0.34	0.34	0.58	0.58
80%	0.52	0.51	1.08	1.08
90%	1.33	1.33	1.45	1.45
95%	1.68	1.68	1.68	1.68
98%	1.88	1.88	1.88	1.88

Figura 3-21

Costul mediu la scriere, măsurat în număr de accese la disc, nu depăşeşte valoarea doi (pentru scrierea blocului de date şi a blocului de paritate) plus costul mediu de curăţire per scriere (de remarcat că pentru RAID 3+1 la o încărcare de 50% costul a fost nul: a fost cazul în care s-a găsit o secţiune cu trei blocuri *free*). Comparând acestea cu un sistem fără *VSP*, în care fiecare scriere (de asemenea a unui singur bloc) necesită patru accese la disc, există întotdeauna o îmbunătăţire a performanţelor. Totuşi, rezultate de un acces la disc per scriere au fost obţinute în cazul 3+1 la o încărcare de până la 80%, iar în cazul 7+1 la o încărcare de până la 70%.

3.6.3.2.3 Comparaţie VSP cu LFS

După cum s-a menţionat anterior, un *LFS* este o abordare cunoscută pentru reducerea întârzierii întâlnită la operaţia de scriere pe discuri. Deşi metodele *LFS* nu au fost iniţial proiectate pentru matrice de discuri, ele se aplică la fel de bine în acest context şi permit ca datele să fie scrise în unităţi de secţiuni: folosind o matrice de discuri cu secţiuni conţinând *N* blocuri de date şi unul de paritate, scrierile pe disc sunt păstrate într-un *buffer* până când se adună minim *N* blocuri şi, apoi, se scriu într-unul sau în mai multe secţiuni *free*. Pentru simplitate vom neglija problemele care pot apărea cu privire la pierderea de date într-un *buffer* parţial plin (în practică, aceste *buffer*-e trebuie scrise pe disc după un anumit timp) şi vom presupune că datele sunt întotdeauna scrise în unităţi de secţiuni.

Folosind această abordare, scrierea a *N* blocuri necesita *N+1* accese la disc (unul pentru fiecare bloc de date şi unul pentru blocul de paritate). Totuşi, acesta poate fi atins prin reorganizarea periodică a segmentelor matricei de discuri pentru a genera matrice conţinând secţiuni *free*, astfel că pentru a compara costul scrierilor, costul reorganizării trebuie luat în calcul.

Segmentele care vor fi reorganizate sunt în mod tipic alese drept cele cu cea mai mică încărcare sau după o funcţie care combină nivelul de încărcare cu vârsta presupusă a datelor în fiecare segment. Indiferent de metoda folosită la selecţia segmentului, va exista o încărcare medie pentru astfel de segmente, să o notăm *u*, şi setul de segmente care trebuie reorganizat va conţine un număr total de secţiuni, să spunem *M*. Folosind metoda de copiere şi compactare, reorganizarea decurge după cum urmează: blocurile *in-use* din toate cele *M* secţiuni sunt citite pentru un total de *uMN* citiri de disc şi, apoi, scrise cu blocurile *in-use* împachetate în secţiuni. Folosind o aproximare continuă, aceasta necesită un total de *uM(N+1)* scrieri (de reamintit că blocul de paritate

este scris pentru fiecare secţiune în faza de scriere). Aceasta conduce la *(1-u)M* secţiuni *free* cu un cost de *uM(2N+1)* accese la disc. După reorganizare, *(1-u)MN* blocuri pot fi scrise în secţiuni *free*, ceea ce necesită *(1-u)M(N+1)* accese la disc ţinând cont şi de scrierea blocului de paritate. Fie C_{LFS} costul scrierii folosind această metodă, definit ca media numărului total de operaţii I/O pentru un bloc scris. Din cele prezentate anterior deducem:

$$C_{LFS} = \frac{uM(2N+1)+(1-u)M(N+1)}{(1-u)MN} = \frac{N+uN+1}{(1-u)N}$$

Acum, să presupunem aceleaşi segmente cu *M* secţiuni şi acelaşi factor de încărcare *u* selectate pentru curăţire şi, ulterior, realocate scrierii într-un sistem VSP. Spre deosebire de abordarea LFS, aici nu există un proces de reorganizare a datelor. În schimb, blocurile protejate *free* din toate astfel de segmente sunt convertite în blocuri neprotejate, iar ulterioarele scrieri se referă la aceste locaţii. Fiind dată o secţiune particulară, să presupunem că încărcarea ei este de *u'*. Deci, există *u'N* blocuri *in-use* şi *(1-u')N* blocuri protejate *free* în aceasta secţiune. Presupunem că după curăţire noul bloc de paritate nu va fi scris pe disc până la prima scriere în secţiune. Prin urmare, costul curăţirii este numărul de citiri de pe disc necesare pentru a recalcula paritatea, care este *min(u'N, 1+(1-u')N)* pentru *u'<1* (pentru *u'=1* costul este zero, deoarece nu există blocuri *free* pentru curăţat). Din păcate, această funcţie nu este liniară în *u'*, ceea ce înseamnă că pentru costul curăţirii sunt necesare informaţii suplimentare asupra distribuţiei lui *u'*. Totuşi, putem obţine marginea superioară a costului prin:

- neţinând cont de cazul special *u'=1* şi, în schimb, folosind *min(u'N, 1+(1-u')N)* drept costul curăţirii pentru toate secţiunile;
- folosind banalele inegalităţi *a>= min(a,b)* şi *b>=min(a,b)*.
- presupunem că, în cazul în care costul curăţirii pentru fiecare secţiune având încărcarea *u'* ar fi *u'N*, costul total ar fi *uMN*. Similar, când costul ar fi întotdeauna *1+(1-u')N*, costul total ar fi *M+(1-u)MN*. De vreme ce acestea două sunt marginile superioare ale costului total, acest cost este mai mic sau egal cu *Mmin(uN, 1+(1-u)N)* citiri de pe disc.

Apoi, după curăţire, să considerăm scrierea în blocurile neprotejate dintr-o secţiune cu încărcarea *u'* a unei secţiuni cu *(1-u')N* blocuri neprotejate. Bufferizând scrierile precum în cazul LFS şi incluzând scrierea blocului de paritate, aceste operaţii necesită *1+(1-u')N* scrieri pe disc pentru *u'<1* (dacă *u'=1*, atunci nici o scriere nu poate avea loc în

această secţiune). Scrierea tuturor celor *(1-u)MN* blocuri neprotejate rezultate din curăţire necesită cel mult *M+(1-u)MN* scrieri pe disc. Rezultă, deci, costul mediu pentru scriere C_{vsp} trebuie să satisfacă relaţiile următoare:

$$C_{VSP} \le \frac{M\min(uN,1+(1-u)N)+M+(1-u)MN}{(1-u)MN} = \frac{\min(N+1,2+2(1-u)N)}{(1-u)N}$$

Pentru a compara aceste două metode, să considerăm raportul C_{VSP}/C_{LFS}. Avem următorul rezultat:

$$\frac{C_{VSP}}{C_{LFS}} \le \frac{\min(N+1;2+2(1-u)N)}{N+uN+1} \le \frac{N+1}{N+uN+1} \le 1$$

Se observă că acest cost la scriere, măsurat în număr de operaţii I/O per bloc scris, este întotdeauna mai mic folosind metoda VSP (neţinând cont de cazul limita *u=0* în care toate secţiunile sunt *free*, în care costurile de scriere sunt evident egale). Marginea superioară a raportului Cvsp/Clfs dat anterior este arătată în Figura 3-22 în funcţie de *u*, pentru matricele de discuri 3+1 şi 7+1. De remarcat că îmbunătăţirile cresc o dată cu creşterea lui *u*. Pentru mărirea factorului de încărcare se poate deduce din relaţiile anterioare următoarea relaţie:

$$\lim_{u\to 1}\frac{C_{VSP}}{C_{LFS}} \le \frac{2}{2N+1}$$

Acest rezultat este interesant deoarece metoda LFS este cunoscută a da rezultate slabe la un factor de încărcare ridicat. În practică este de aşteptat ca performanţele relative ale metodelor VSP, în comparaţie cu metodele LFS, să fie mai bune decât cele sugerate anterior, ţinând cont de faptul că expresiile derivate din raportul costului mediu la scriere între metodele VSP şi LFS sunt referitoare la marginile superioare, în practică, însă, raportul real fiind mai mic.

În general, presupunerile simpliste dau un avantaj îndoielnic metodelor LFS (exemplu: se presupune că metodele LFS întotdeauna scriu datele la un anumit nivel de ocupare, deşi, după cum s-a remarcat anterior, în practică *buffer*-ele parţial încărcate pot fi forţate să scrie datele pe disc chiar la un nivel de ocupare mai mic decât cel considerat).

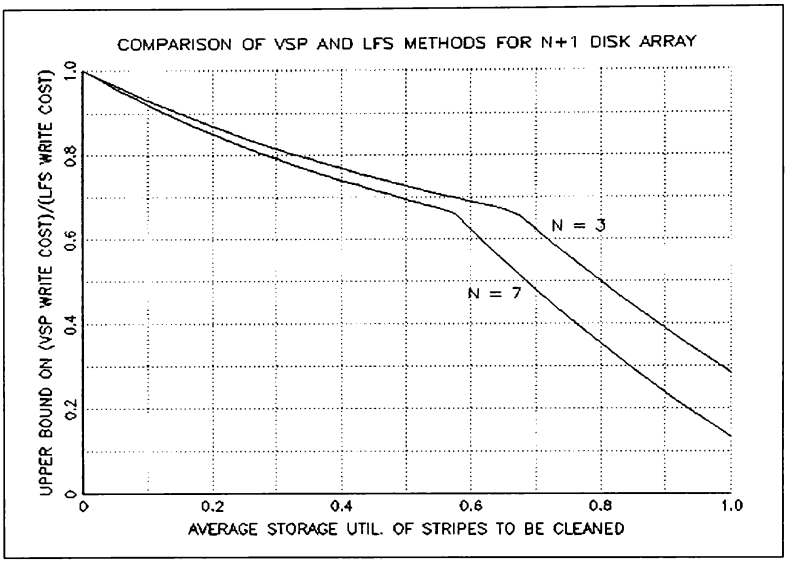

Figura 3-22

3.6.3.2.4 Costul recuperării datelor de pe discul căzut

În acest capitol, vom considera costul recuperării blocurilor de date şi de paritate în cazul defectării unui singur disc din aria de N+1 discuri. O analiză a costului recuperării discului defect este îngreunată de faptul că, în practică, pentru a grăbi recuperarea, mari cantităţi de date (chiar cilindri întregi) trebuie citite de pe fiecare disc nedefectat în acelaşi timp. Totuşi, vom estima aici munca totală necesară recuperării doar prin numărarea blocurilor care trebuie citite de pe fiecare disc nedefectat pentru a recupera un bloc ales la întâmplare de pe discul defectat. Fără VSP şi cu grupuri de paritate implementate în secţiuni, acest număr este întotdeauna N, deoarece un bloc trebuie citit de pe fiecare celelalte discuri.

Să presupunem acum că folosim o schemă cu VSP. Atunci, o parte p din toate blocurile de date este protejată, iar restul blocurilor sunt neprotejate. Pentru a simplifica următoarele analize, să presupunem că fiecare bloc de date este protejat cu probabilitatea p, independent de starea de protecţie a celorlalte blocuri de date.

Există cel puţin două moduri în care poate avea loc recuperarea, depinzând de detaliile modelului sistemului, precum şi de modul de defectare al discului. Într-unul din cazuri, nici o informaţie de protecţie

alta decât cea stocată în aria de discuri nu este disponibilă şi, de asemenea, informaţia *in-use* păstrată de *storage manager* nu este disponibilă. Totuşi, s-ar putea ca informaţiile asupra stării de protecţie a blocurilor de disc să fie disponibile altundeva, de exemplu într-o hartă păstrată în *storage manager*. În acest caz, informaţiile *in-use* ar fi probabil disponibile, dar pentru simplitate vom examina cazul recuperării folosind doar informaţia de protecţie. Ambele cazuri vor fi tratate mai jos, în primul rând analizând cazul în care nu este disponibilă informaţie de protecţie (sau *in-use*).

În primul rând, fiind dat un bloc ales arbitrar de pe discul defectat, deoarece blocurile de paritate sunt egal distribuite pe toate discurile, acest bloc are probabilitatea de a fi un bloc de paritate de *1/(N+1)* şi de date de *N/(N+1)*.

Dacă blocul de pe discul defect este un bloc de paritate, atunci, dacă informaţia asupra stării blocurilor de date din secţiune nu se găseşte disponibilă în alt loc, este de presupus că fiecare bloc de date poate fi *in-use*, necesitând citirea tuturor celor *N* blocuri de date de pe celelalte discuri pentru a calcula noul bloc de paritate (alternativ, dacă informaţia de protecţie este disponibila, numărul de blocuri de date necesar pentru a calcula noul bloc de paritate ar fi *pN*).

În caz contrar, blocul de pe discul defectat este un bloc de date. Presupunând că informaţia de protecţie este disponibilă doar din blocul de paritate al secţiunii, cel puţin un bloc (de exemplu, blocul de paritate) trebuie citit pentru a recupera informaţia de protecţie (dacă informaţia de protecţie este disponibilă în alt loc, această citire nu este necesară). Recuperând informaţia, blocul va fi neprotejat cu o probabilitate de *1-p,* caz în care nici o acţiune ulterioară nu e necesară. Blocul este protejat, cu probabilitatea *p*, caz în care numărul cerut de blocuri de date pentru a efectua recuperarea informaţiilor este *p(N-1)*.

Combinând rezultatele anterioare, numărul de blocuri necesare pentru a recupera blocul considerat de pe discul defectat este:

$$N_h = \frac{N}{N+1} + \frac{N(1-p)}{N+1} + \frac{Np}{N+1}(1+p(N-1)) = N\left(\frac{2+p^2(N-1)}{N+1}\right)$$

Alternativ, să presupunem că informaţia de protecţie este disponibilă pe o sursă, alta decât blocurile de paritate din aria de discuri (dar nu informaţia *in-use*). Repetând analizele anterioare, considerate pentru acest caz, numărul de blocuri necesare pentru a recupera blocul considerat de pe discul defectat este:

$$N_b = \frac{pN}{N+1} + \frac{N(1-p)0}{N+1} + \frac{Np}{N+1}(1+p(N-1)) = N\left(\frac{2p+p^2(N-1)}{N+1}\right)$$

Să presupunem, de exemplu, că $N=7$ şi că jumătate din blocurile de date din aria de discuri sunt protejate. În cazul în care informaţia de protecţie este disponibilă doar în blocurile de paritate din aria de discuri, vom găsi că $(2+0.5^2 \times 6)/8=43.75\%$ blocuri trebuie citite pentru a recupera fiecare bloc defectat în comparaţie cu recuperarea în cazul RAID 5. În mod similar, dacă informaţia de protecţie este disponibilă în alt loc, doar $(2 \times 0.5 + 0.5^2 \times 6)/8 = 31.25\%$ blocuri trebuie citite. Reducerea costului recuperării, comparat cu RAID 5 folosind anterioarele două metode este arătată în Figura 3, ca o funcţie de raportul blocurilor de date protejate, pentru ariile de discuri 3+1 şi 7+1.

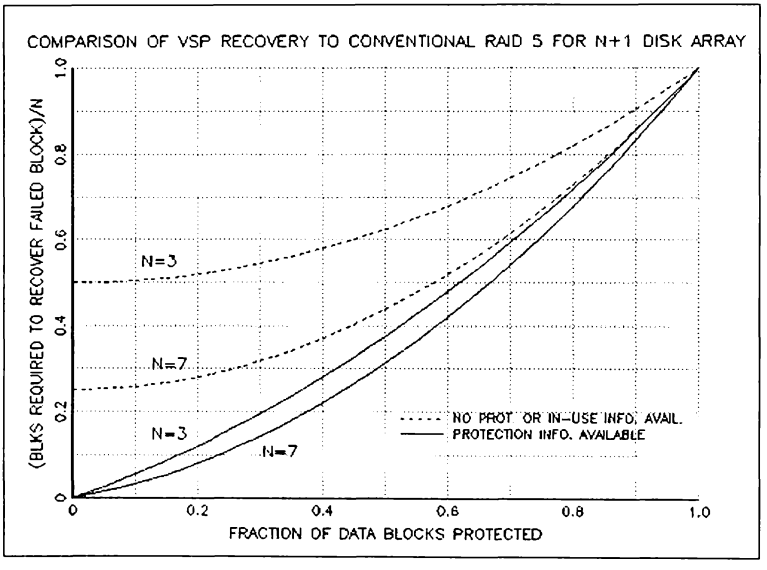

Figura 3-23

Dacă atât informaţia de protecţie, cât şi cea *in-use* sunt disponibile (de exemplu, din *storage manager*) şi sunt folosite în timpul recuperării, costul recuperării devine şi mai mic (doar dacă nu cumva aria de discuri se află în starea în care toate secţiunile sunt curate). O astfel de analiză implică adăugarea unui număr de presupuneri cu privire la numărul şi distribuţia blocurilor protejate *free*. Deoarece aceste presupuneri sunt dependente de felul în care lucrează *storage manager*-ul şi de algoritmii utilizaţi de acesta, aici vom considera o situaţie mai simplificată (de remarcat că rezultatele anterioare ale costului recuperării când

informaţia de protecţie este disponibilă în alt loc este marginea superioară a costului recuperării în cazul unor metode care folosesc atât informaţia de protecţie, cât şi cea *in-use*). Prin urmare, în practică, este de aşteptat ca acest cost să fie mai mic în acest caz decât cel indicat de rezultatele anterioare. De asemenea, este de remarcat faptul că asemenea metode nu doar recuperează blocurile de pe un disc defectat, dar (cu nici un cost suplimentar, în timpul procesului de recuperare) au şi un efect lateral, de curăţire a secţiunilor.

3.6.3.2.5 Concluzii

Am prezentat o extensie a arhitecturii RAID 5 în care proporţia informaţiei de protecţie pentru fiecare secţiune poate varia. Folosind VSP, informaţia de paritate este menţinută pentru subseturi de blocuri de date în fiecare secţiune. Dacă paritatea este păstrată pentru un bloc de date dat, se spune că este protejat; altfel este neprotejat. Presupunem că, în practică, toate blocurile de date *in-use* trebuie să fie protejate. Blocurile de date *free* (nealocate) sunt iniţial neprotejate. Cum scrierile au loc în blocuri de date neprotejate, blocurile devin protejate. Dacă data stocată într-un bloc *in-use* (protejat) este ştearsă sau mutată, starea blocului devine protejat *free*.

Scrierile pe disc la o valoare mai mică decât granularitatea secţiunii sunt probleme de performanţă cunoscute pentru arhitecturile RAID 5: scrierea unui singur bloc poate necesita până la patru operaţii I/O. Prin scrierea în blocuri neprotejate, această problemă se reduce, deoarece scrierea unui singur bloc necesită cel mult trei operaţii I/O în acest caz. Dacă scrierea se face intr-un bloc deja alocat, este necesară realocarea scrisului. Când se realocă o scriere, noua locaţie devine protejată *in-use* şi anterioara locaţie este protejată *free*. Totuşi, pentru a menţine o cantitate de blocuri neprotejate, este necesară o curăţare periodica prin care blocuri protejate *free* sunt convertite în blocuri neprotejate.

Simulările au fost făcute de către un *storage manager* „minimal“ care efectuează următoarele:

- găseşte şi curăţă cea mai puţin utilizată secţiune;
- realocă toate scrierile ulterioare a câte unui singur bloc aceasta secţiune până când aceasta este plină.

S-a descoperit că există întotdeauna o îmbunătăţire în costul scrierii (măsurat în media numărului total de operaţii I/O per scriere) comparat cu modelul convenţional de RAID 5. Totuşi, îmbunătăţiri substanţiale au apărut doar pentru încărcări mici, până la medii. De exemplu, o

economie de mai mult de un acces la disc per scriere a fost obţinută pentru matricele de discuri 3+1 pentru încărcări de până la 80%, iar pentru matricele de discuri 7+1 pentru încărcări de până la 70%. În practică, este de aşteptat ca performanţele să fie simţitor mai bune, deoarece scrierile nu sunt aleatoare şi uniform distribuite (exemplu: seturi de blocuri dintr-un fişier dat pot fi citite şi scrise împreună).

Un mod alternativ de a reduce costul scrierii este de a forţa scrierea la o anumită granularitate a secţiunii. Un mod de a face acest lucru este utilizarea LFS. Însă, aceasta necesită reorganizarea periodică a discului (cunoscută în literatura LFS drept curăţire) pentru a genera noi zone libere. Folosind câteva presupuneri simple, rezultatele analitice au fost obţinute comparând costul mediu de scriere pentru metodele VSP şi LFS. S-a arătat că metodele VSP au un cost mediu de scriere mai scăzut în toate cazurile, cu îmbunătăţiri mărite la creşterea încărcării. Acest rezultat este interesant deoarece este ştiut că metodele LFS prezintă deficienţe ale performanţei la o încărcare ridicată.

Costurile recuperării informaţiei au fost analizate pentru două cazuri. Într-unul dintre acestea, nici o informaţie alta decât cea aflată pe discurile funcţionale din aria de discuri nu este disponibilă. În al doilea caz, toată informaţia de protecţie (dar nu informaţia *in-use*) este disponibilă în alt loc (de exemplu, din sistemul *storage manager*-ului). În ambele cazuri, costul recuperării informaţiei a fost mai mic decât cel al sistemelor convenţionale RAID 5, cu o reducere şi mai mare o dată cu scăderea numărului de blocuri protejate. De exemplu, cu 50% blocuri de date protejate, costul recuperării a fost de 44% în raport cu sistemele convenţionale RAID 5 în primul caz, şi cu 31% în cel de-al doilea. Dacă şi informaţia *in-use* se foloseşte pentru recuperarea informaţiilor, costul va fi şi mai mic şi, pe deasupra, ca un efect lateral benefic al recuperării, un anumit număr de secţiuni vor fi curăţate prin procesul de recuperare.

Rezumând, permiţând blocului de paritate să protejeze un subset de blocuri de date dintr-o secţiune este o idee simplă care poate duce la micşorarea problemelor întâlnite la sistemele RAID 5 în cazul scrierilor mici. În general, aceste modele necesită relocarea scrierilor şi curăţirea periodică. În multe sisteme aceasta se potriveşte foarte bine cu sistemul de fişiere existent şi/sau programele de aplicaţii. De exemplu, în unele sisteme, din motive care ţin de recuperarea informaţiilor, modificările unui fişier existent sunt efectuate întâi prin scrierea noii versiuni a fişierului (în spaţiul liber de pe disc), şi abia apoi ştergând versiunea anterioară a fişierului (eliberând spaţiul ocupat de vechea versiune). Similar, reorganizarea periodică a discului este făcută astăzi în mod

obişnuit pentru a reduce fragmentarea, a recupera blocuri pierdute din cadrul blocului de fişiere s.a.m.d.

3.6.3.3 RAID 5/ NRP

Matricele de discuri au fost propuse pentru a asigura un spaţiu mare de stocare de date, asigurând o înaltă fiabilitate la un cost redus. În organizarea RAID (matrice redundante de discuri independente) sunt *N+1* discuri, acestea conţinând un bloc de paritate pentru *N* blocuri de date, fiecare fiind stocat pe un disc diferit. (În acest subcapitol vom lua în consideraţie specificaţiile RAID 5).Vom numi setul de blocuri de date, împreună cu blocul de paritate, grup de paritate.

Dacă un disc se defectează, fiecare bloc de date de pe acest disc se poate reface din celelalte *N* blocuri aflate în grupul de paritate.

În orice fel, aceasta poate duce la o substanţială degradare de performanţă în timpul refacerii, deoarece discul defect poate dubla încărcarea pe discurile rămase, presupunând că fiecare disc este egal încărcat.

Acest inconvenient al RAID-ului tradiţional poate fi rezolvat dacă se utilizează conceptul *clustered* RAID (RAID grupat) propus de Muntz şi Lui.

Clustered RAID este un sistem RAID avansat în care grupurile de paritate de dimensiune *G* se întind peste *cluster*-ele discurilor de dimensiune *C (C>G)*.

Într-un astfel de sistem încărcarea datorată discului defect este distribuită eventual peste toate cele *C–1* discuri rămase, rezultând o degradare de performanţă mai mică. Un *cluster* de dimensiune mare poate fi utilizat pentru a reduce timpul de refacere şi reconstrucţie a informaţiilor de pe discul defect, ceea ce conduce la o performanţă mai mare.

Cu toate acestea mărirea dimensiunii *cluster*-ului are un efect negativ prin mărirea ratei la al doilea eşec. Analizele arată că acest efect poate fi compensat de un timp de refacere mic, conducând la o însemnată îmbunătăţire a timpului sistemului defect.

Deşi sunt multe metode de refacere a informaţiei de pe discul defect, nici una nu asigură o corespondenţă între blocurile de date şi adresele de disc sau adresele blocurilor de date şi adresele grupurilor de paritate pentru *C* şi *G* date.

O schemă simplă care foloseşte toate combinaţiile de *G* discuri luate câte *C* este greu de implementat, pentru că sunt *(C/G)* alegeri diferite şi o tabelă care să reţină adresele blocurilor poate ocupa un spaţiu foarte mare. În plus, spaţiul se poate mări dacă numărul grupurilor de paritate din aria de discuri este substanţial mai mic decât *(C/G)* această schemă de distribuţie nu asigură o încărcare aleatoare.

Considerăm *C=6* şi *G=3*. Presupunem că avem numai cinci grupuri de paritate; în acest caz o harta a grupurilor de discuri poate fi:

(1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 2, 6), (2, 3, 4). Este clar că nu se asigură o încărcare aleatoare.

În continuare, prezint un algoritm rapid de distribuire a grupurilor de paritate de dimensiune *G* pe cele *C* discuri. Muntz şi Lui notează că aceasta poate fi tratată ca o problemă de proiectare a unui bloc incomplet balansat. Holland şi Gibson sugerează scheme bazate pe descrierea incompleta a blocurilor când acestea sunt disponibile. Convenim să egalizăm încărcarea de pe fiecare disc prin maparea aleatoare a fiecărui grup. Maparea este creată pentru fiecare grup de paritate bazându-ne pe permutări aleatoare. Această schemă este aplicată oricărei combinaţii de *G şi C (G <= C)*. Cu toate acestea atât teoria cât şi simularea practică arată că este o metodă rapidă şi se ocupă foarte puţin spaţiu.

După ce am stabilit caracteristicile conceptului *clustered* RAID, ne vom axa pe prezentarea modelului analitic, care este tratat în continuare şi pe evaluarea performanţelor.

Procesele de citire şi de scriere din aria de discuri sunt destul de complicate, astfel fiecare acces logic necesită un număr variabil de citiri şi scrieri fizice, depinzând de faptul că se face o citire logică sau o scriere logică şi de faptul ca operaţia s-a efectuat cu succes sau nu. Dacă accesul se face la un disc defect depinde de configuraţia datelor de pe acel disc şi de amplasarea blocului de paritate. În plus, în timpul operaţiei de refacere, procesele de reconstrucţie a informaţiei modifică nivelul de încărcare şi, dacă se foloseşte o metodă complexă ca „reconstrucţia cu redirectarea citirilor", încărcarea sistemului poate fi foarte mare.

Modelul nostru analizează timpul de răspuns la citire şi scriere într-un caz normal şi în cazul când se defectează un disc, precum şi timpul de refacere a informaţiilor, urmărind efectele prin variaţia raportului *C/G* şi încărcarea externă. Modul normal de funcţionare este modelat utilizând o coadă *M/G/1*, în timp ce efectul unui singur disc defect şi procesul de

reconstrucţie a informaţiei este captat de o coadă de tip *permanent customer*. Acesta este un model analitic larg utilizat, fiind general şi aplicabil şi altor probleme.

Rezultatele sunt validate prin comparare cu datele obţinute din simulare. Analiza noastră arată că modelul *clustered* RAID este semnificativ mai tolerant la defectarea unui disc decât modelul RAID clasic. Atât timpul de refacere cât şi degradarea de performanţă din timpul refacerii datelor sunt substanţial reduse în modelul *clustered* RAID; mai mult decât atât, acest avantaj poate fi obţinut prin utilizarea unui raport C/G mic.

3.6.3.3.1 Construirea distribuţiei bazate pe NRP

Considerăm o matrice de C discuri gestionate de un controler, cu un grup de paritate format din G blocuri ($G <= C$, $G-1$ blocuri de date şi un bloc de paritate). Fiecare bloc este plasat pe un disc diferit, astfel oricare bloc aflat pe discul defect poate fi refăcut din celelalte blocuri aflate în grupul de paritate. În continuare, presupunem că un bloc este o pistă de disc.

Presupunem că sunt B blocuri disponibile pe disc. Din punctul de vedere al sistemului de operare, întreaga matrice de discuri apare ca un singur disc logic, cu blocurile numerotate $0,1,..., BC-1$; aceste blocuri includ şi blocul de paritate. Subseturile $\{0,1,..., G-1\}$, $\{G,G+1, ..., 2G-1\}$, ... formează grupuri de paritate, iar ultimul bloc din fiecare grup este blocul de paritate.

Aceasta configuraţie logică este oglindită într-o matrice fizica adresată astfel: Address(i) = (Disc(i), LocalBloc(i)), unde indicele i indică numărul blocului LocalBloc(i) aflat pe discul Disc(i).

Această schemă de adresare trebuie să îndeplinească unele condiţii:

- trebuie ca maparea numerelor dintr-un grup de paritate să conţină discuri diferite;
- blocurile de paritate trebuie să fie reactualizate ori de câte ori un bloc aflat în grupul de paritate este modificat, acestea trebuie să fie distribuite uniform pe toate discurile;
- pagina de date cerută, aflată pe un disc defect cauzează citiri de pe celelalte $G-1$ discuri rămase. Pentru a echilibra încărcarea din timpul defectului trebuie ca blocurile de pe fiecare disc să fie uniform distribuite;
- adresele trebuie să fie uşor de calculat.

În continuare se prezintă o schemă de adresare simplă bazată pe permutări aleatoare. Pentru simplitate, presupunem că C este par şi că G divide C, dar mai târziu se va înlătura această restricţie.

Fie $P_0, P_1, \ldots$ permutări aleatoare ale $(0,1,\ldots, C-1)$ unde P_n mută j în $P_{n,j}$. Fie $n=\lfloor i/C \rfloor$ şi $j=i \bmod C$, atunci schema de mapare asociază blocului i adresa Address(i)≡(Disk(i), LocalBlock(i))=($P_{n,j}$, n). Este uşor de observat că următoarea schemă satisface condiţiile 1), 2) şi 3). Dacă G divide C nici un grup de paritate nu este obţinut din permutări distincte şi astfel nici care două blocuri din acelaşi grup de paritate, nu se pot afla pe acelaşi disc.

Cum fiecare permutare este aleatoare, blocul de paritate din fiecare grup, este distribuit aleator pe discuri şi fiecare grup de paritate este mapat pe un set de discuri care sunt neuniform selectate din cele (C/G) subseturi de discuri.

Problema rămasa ar fi aceea daca $P_{n,j}$ se poate calcula eficient. O variantă ar fi de a calcula P_n de fiecare data când este nevoie de $P_{n,j}$. Aceasta este ineficientă atunci când C este mare şi, prin urmare, se propune calcularea lui $P_{n,j}$ în $O(\log_2 C)$ paşi utilizând metoda *Thorp's shuffle*.

3.6.3.3.2 Algoritmul „Thorp's shuffle"

Vom descrie acest algoritm exemplificând pe un teanc de cărţi de joc:

- Se taie cărţile în două jumătăţi egale A şi B
- Se extrage la întâmplare o carte din teancul A şi una din teancul B, apoi se extrage cea mai de jos carte din primul teanc şi peste ea se pune cea mai de jos carte din al doilea teanc.
- Se repetă pasul al doilea până se termină cele două teancuri.

Exista $C=2N$ cărţi numerotate $0,1, \ldots, 2N-1$, „amestecarea lui Thorp" mută cartea j în poziţia $T(j)$, unde U_j este un bit aleator şi

$$T(j) = 2j+U_j \quad \text{pentru } j < N \qquad (1)$$

$$T(j+N) = 2j+1-U_j \quad \text{pentru } j < N \qquad (2)$$

Această amestecare se poate repeta până când cărţile sunt permutate aleator, astfel se generează a n–a permutare aleatoare P_n, începând cu permutarea iniţială şi aplicându-se K amestecări, utilizând o secvenţă de N biţi aleatori $U^{(n)}_{m1}$, $U^{(n)}_{m2}$, ... , $U^{(n)}_{mN}$ pentru m amestecări. Pentru a găsi $P_{n,j}$ vom urmări poziţia cărţii j după fiecare amestecare; poziţia

după K amestecări este $P_{n,j}$. Experimentul se face pentru $C<=32$ şi $U^{(n)}_{mj}$, d variind valorile lui G indicând $K=2log_2(C)$ amestecări aleatoare ale permutărilor suficient pentru a evalua încărcarea datorată discului defect. În continuare, vom pune problema generării eficiente a biţilor aleatori.

3.6.3.3.3 Generarea secvenţei de biţi aleatori

Fiecare permutare din metoda de mai sus necesită K amestecări şi fiecare amestecare foloseşte N biţi aleatori. Este uşor de văzut că $U^{(n)}_{mj}$ este un număr de biţi aleatori $NKn+N(m-1)+j$.

Atribuim $U^{(n)}_{mj} = [2X_{NKn+N(m-1)+j} / M]$, unde X_i este un număr pseudo-aleator uniform distribuit în $[0,M)$.

Dacă X_i ar fi generat printr-o recurentă linear-congruenta $X_{i+1}=(aX_i+c) mod M$, atunci:

$$X_{i+k} = (a_k X_i + c_k) \bmod N \qquad (3)$$

unde $a_k \equiv a^k \ mod\ M$ şi $C_k \equiv \left(C\left(a^k - 1\right)/(a-1)\right) mod\ M$

Scriem k în baza b: $k=k_0+k_1 b+ \ldots + k_i b^r$. Acum X_k se poate calcula în $r = [\log_b k]$ paşi utilizând (3),

$$X_{k_0+k_1 b+\ldots+k_i b^r} = \left(a_{k_i b^r} X_{k_0+k_1+\ldots+k_i b^r} + c_{k_i b^r}\right) \bmod M \qquad i = 0,\ldots,r \qquad (4)$$

Atunci acestea sunt blocurile BC în aria de discuri şi se arată uşor că

$k<=BCk/2$ şi, prin urmare, este nevoie să se stocheze numai valorile a_i şi c_i pentru $i=pb^n$, $p=1,2,\ldots, b-1$ şi $n=0,1,2,\ldots,[log_b(BCK/2)]$

Stocarea necesită pentru aceasta $O(b \log_b (BCK/2))$. Alegând o

valoare pentru b potrivita (*ex:* $b=100$), X_k poate fi calculat rapid, fără cerinţe mari de stocare.

Algoritmul *Thorp's shuffle* caută adresele de disc $Disk(nC+j)=P_{n,j}$ utilizând $K=2 \log_2 C$ numere pseudo-aleatoare $X_{q1}, X_{q2},\ldots, X_{qK}$. pentru subsecvenţa $X_{NKn}, X_{NKn+1},\ldots, X_{NKn+1}$. Folosim această metodă o singură dată pentru a calcula X_{q1}, iar $X_{q2},\ldots, X_{qK}$ se calculează direct din (3):

$$X_{q_{i+1}} = X_{q_i+(q_{i+1}-q_i)} = \left(a_{(q_{i+1}-q_i)} X_{q_i} + b_{(q_{i+1}-q_i)}\right) \bmod M$$

Atunci $q_{i+1} - q_i <= 2N=C$, X_{qi}; se poate afla într-un singur pas dacă se antecalculează a_k şi b_k pentru $k<=C$.

Timpul total pentru calcularea adreselor de disc pentru orice bloc folosind această metodă este $O(\log_b B+\log_b C)$ şi costul de memorare a informaţiei este $O(\log_b B+C \log_b C)$.

În prezentarea anterioară am impus ca G să fie un multiplu de C şi C să fie par, pentru simplitate. Fiecare dintre restricţii se poate înlătura.

Impunem ca G să dividă C ca să se respecte condiţia (1) ca nici un grup de paritate să nu fie spart peste permutări distincte.

Pentru un G generalizat nu folosim o permutare numai o dată ci de $R=LCM(C,G)/C$ ori consecutiv. Astfel, fiecare permutare distribuie un set de blocuri RC consecutive, astfel nici un grup de paritate nu este spart peste permutări distincte, satisfăcând condiţia (1).

Observaţie: C este necesar să fie par pentru a se putea aplica *Thorpe's shuffle*, iar dacă C nu este par se modifică *Thorpe's shuffle* alegând $N=[C/2]$

şi $U_{N-1}=0$ totdeauna. Simularea arată că aceste modificări creează permutări aleatoare satisfăcătoare.

3.6.3.3.4 Un model pentru RAID5/NRP

Considerând aria de discuri formata din C discuri, cu un grup de paritate de dimensiune G. Traficul este dat de distribuţia neuniformă a discurilor, sosirile cererilor de citire şi scriere de pe fiecare disc trebuie sa fie în conformitate cu procesele date de metoda *Poisson*, procese care au ratele de sosire λ_r şi $\lambda_w; \lambda = \lambda_r + \lambda_w$. Unitatea de date cerută se presupune uniform distribuită pe discuri.

Fiecare disc are o coada FIFO pentru cereri. Timpul de servire pentru o cerere fizică este compus din timpul necesar căutării, latenţa şi timpul necesar transferului de date. Timpul de căutare se poate aproxima astfel:

$Seek = a + b\sqrt{\Delta}$, unde Δ este fracţiunea pistei traversate de capul de disc. Latenţa se presupune uniform distribuită în $(0,t_r)$, unde t_r este timpul de rotaţie al discului (valorile considerate se pot vedea în Figura 3-24). Timpul de transfer pentru o pagină citită este constant t_p.

Parameter	Denoted	Value
arm movement time	a	3ms
seek factor	b	27ms
disk rotation time	t_r	16.8ms
page transfer time	t_p	2ms
page size		4KB
block size		1 track
block transfer time	t_b	16.8ms
disks in cluster	C	16
blocks in group	G	8
pages per track		8
tracks per disk		16,000

Figura 3-24

La scriere procesul este mai complex. Presupunem că dorim să înnoim informaţia din pagina V_0, grupul de paritate conţine paginile V_1,..., V_{G-2} şi P (din blocul de paritate). Dacă noua valoare a paginii V_0 este V_0', atunci noua valoare a lui P este, $P'=P\ XOR\ V_0\ XOR\ V_0'$. Operaţia de înlocuire constă în două operaţii de citire/scriere pe discuri diferite. Prima citeşte V_0 şi scrie V_0', iar a doua citeşte P, calculează P' şi îl scrie.

Timpul de calcul/transfer pentru fiecare din aceste operaţii este t_r+t_p, exceptând cazul când V_0 se află pe discul defect, când noua valoare de paritate se construieşte după formula

$$P' = V_0'\ XOR\ V_1\ XOR\ \ldots\ XOR\ V_{G-2}$$

Întârzierea în acest caz este maximum dintre întârzierile citirilor şi timpul de scriere P'.

Când un disc se defectează, datele cerute de pe el sunt refăcute prin citirea datelor din celelalte blocuri aflate în grupul de paritate.

Presupunem că un proces de refacere reconstruieşte datele de pe discul defect accesând datele de pe discurile valide. Presupunem că acest proces se desfăşoară asincron. Procesul (iniţiat de unitatea de control) citeşte de pe fiecare disc valid un bloc care face parte din grupul de paritate al discului defect şi inserează într-un *buffer* aflat în unitatea de control, pentru al furniza la următoarea cerere direct din coadă. Pe discul disponibil, procesul reface blocul prin aducerea grupului de paritate din *bufferul* unităţii de control.

Vom urmări metoda de reconstrucţie cu redirectarea citirilor.

Înnoirea informaţiilor necesită două scrieri: una în pagina ţintă şi una în pagina corespunzătoare din grupul de paritate. Dacă se încearcă o scriere pe discul defect aceasta se redirectează spre o zonă deja refăcută.

3.6.3.3.5 Analiza performanţei RAID5/NRP în modul de operare normal

Într-o operaţie efectuată cu succes ne interesează întârzierile operaţiilor de citire şi scriere şi rezultatul maxim care se poate obţine. Acesta se poate obţine prin modelarea fiecărui disc printr-o coadă *M/G/1*, procesul dorit se obţine prin aplicarea transformatei *Laplace* şi *Poisson* distribuţiei timpului de căutare, latenţei şi timpului de transfer astfel:

$$Seek^*(s) = E\left[e^{-sSeek}\right] = E\left[e^{-s(a+b\sqrt{|\Delta|})}\right] =$$

$$= \frac{(24 + 24bs + 8b^2s^2)e^{-(a+b)s} - (24 - 4b^2s^2)e^{-as}}{b^4s^4}$$

$$Latency^*(s) = E\left[e^{-sLatency}\right] = (1 - e^{-st_r})/(t_r s)$$

$$Transfer_r^*(s) = E\left[e^{-sTransfer}\right] = e^{-st_r}$$

$$Transfer_w^*(s) = E\left[e^{-sTransfer_w}\right] = e^{-s(t_r + t_F)}$$

Prin urmare LT aplicată distribuţiei timpului total este:

$$B^*(s) = E\left[e^{-sService}\right] =$$

$$= Seek^*(s)Latency^*(s)(\lambda_r Transfer_r^*(s) + \lambda_w Transfer_w^*(s))/(\lambda_r + \lambda_w)$$

LT aplicata distribuţiei timpului de aşteptare în coadă este data de:

$$W^*(s) = E\left[e^{-sW}\right] = (s(1 - \lambda E[Service]))/(s - \lambda(1 - B^*(s)))$$

Întârzierea totală pentru citirile de pe discul *i* este suma dintre timpul de aşteptare şi timpul necesar citirii şi, analog, întârzierea la scriere.

Cum traficul este neuniform distribuit pe discuri, întârzierea la scriere este egală cu cea la citire, astfel întârzierea locală la citire este egală cu cea la citirea din matrice şi întârzierea la scriere în matrice este dublă faţă de întârzierea scrierii locale. Pentru a obţine rezultatul maxim pentru fiecare cerere externa de modificare de date se fac două scrieri locale. Dacă ratele de sosire a citirilor şi a scrierilor în matrice sunt Λ_r şi Λ_w, respectiv pentru discuri Λ_r/C şi Λ_w/C. Aplicând restricţia $\lambda E[Service]<=1$ se obţine rezultatul maxim în matrice.

$$\Lambda_r(a+8b/15+t_r/2+t_p)+2\Lambda_w(a+8b/15+3t_r/2+t_p)\le C\,.$$

3.6.3.3.6 Analiza performanţei RAID5/NRP în modul de operare în avarie

Analiza matricei de discuri în caz de defecţiune, este complicată de interacţiunea dintre accesele externe şi procesul de reconstrucţie *on-line* descris anterior.

Considerăm fiecare disc în parte. Fie T_m timpul de sosire al celei de-a *m*-a cereri de reconstrucţie, $Y_m=T_m - T_{m-1}$ întârzierea pentru a *m-1* cerere de reconstrucţie şi N_m numărul cererilor externe dinaintea sosirii cererii *m*. Dacă $N_m>0$, atunci distribuţiile timpilor de căutare şi de latenţă pentru a *m*-a cerere de reconstrucţie sunt date de (6) şi (7). Dacă $N_m=0$ atunci cererea de reconstrucţie urmează anterioarei cereri de reconstrucţie imediat şi atunci devine uşor de urmărit timpii de căutare şi latenţă împreună. Deoarece fiecare bloc este o pistă şi capul trebuie să schimbe pistele şi apoi să aştepte sosirea începutului fiecărei piste, timpii combinaţi de căutare şi latenţă trebuie să fie egali cu cel puţin timpul unei rotaţii de disc. Totuşi, în timpul unei rotaţii, capul poate căuta de-a lungul unui număr mare de piste pentru presupusele valori ale parametrilor. Prin urmare, timpii de căutare şi latenţă egalează t_r, cu LT:

$$SeekLatency^*(s)=E\left[e^{-sSeekLatency}\right]=e^{-st_r}$$

Rezumând, aceasta este o coadă cu sosiri de tip *Poisson*, cu rata λ şi timpul de servire corespunzător *LT B*(s) şi* un client permanent (procesul de reconstrucţie) care accesează coada din nou îndată ce îşi termină serviciul, cu timpul de servire:

$$B_q=\begin{cases} SeekLatency+BlockTransfer & N_m=0 \\ Seek+Latency+BlockTransfer & N_m>+ \end{cases}$$

Fie Y întârzierea stării uniforme pentru cererile de reconstrucţie şi W timpul de aşteptare al cozii pentru citiri şi scrieri locale. Rezultă următoarele ecuaţii:

$$Y^*(s) \equiv E\left[e^{-sY}\right] = Y^*(\lambda)a(s) + b(s)Y^*(\Phi(s))$$

$$E[Y] = -(pa'(0) + b'(0))/(1 - \Phi'(0))$$

$$W^*(s) \equiv E\left[e^{-sW}\right] = -(Y^*(s) - Y^*(\lambda(1 - B^*(s)))/((s - \lambda(1 - B^*(s)))B^*(s)$$

$$E[W] \equiv W^{*'}(0) = (1 - \lambda B^{*'}(0))E\left[Y^2\right]/(2E[Y])$$

unde:

$$a(s) \equiv BlockTransfer^*(s)(SeekLatency^*(s) - Seek^*(s)Latency^*(s))$$

$$b(s) \equiv Latency^*(s)BlockTransfer^*(s)Seek^*(s)$$

$$p \equiv Y^*(\lambda) = \frac{\prod_{i=0}^{\infty} b(\Phi^i(\lambda))}{1 - \sum_{i=0}^{\infty} a(\Phi^i(\lambda))\prod_{j=0}^{i-1} b(\Phi^j(\lambda))}$$

$$\Phi(s) \equiv \lambda(1 - B^*(s)); \ \Phi^i(s) \equiv \Phi(\Phi^{i-1}(s)): \ \Phi^0(s) \equiv s$$

Valorile momentelor Y se pot obţine prin diferenţiere. Rezultatul acestui disc pentru procesul de refacere este *1/E [Y]*. Întârzierile pentru citiri şi scrieri locale de pe acest disc sunt date prin suma timpilor corespunzători de servire. Totuşi, citirile externe făcute de pe discul defect şi toate actualizările externe pot implica discuri multiple.

Deoarece o parte a citirilor de pe discul defect pot fi redirectate către spaţiul disponibil dacă datele au fost deja refăcute, restul trebuie reconstruite *on-line* prin citirea celor *G-1* grupuri corespunzătoare. Întârzierea efectivă pentru o astfel de citire reprezintă maximul întârzierilor pentru citiri locale de pe cele *G-1* discuri implicate. Deoarece traficul este uniform pe aceste discuri, întârzierile citirilor locale sunt distribuite în mod identic şi media lor μ şi variaţia σ^2 pot fi găsite ca şi în cazurile de mai sus. Pentru simplitate vom aproxima aceste întârzieri ale citirilor, ca valori extreme independente ale variabilelor aleatoare cu media μ şi variaţia σ^2. Deci, maximum acestor $G - 1$ variabile aleatoare este o valoare extremă cu media $\mu + \left(\sqrt{\delta}\sigma/\pi\right)\ln(G-1)$ şi variaţia σ^2.

Acestea ne conduc la o aproximare pentru întârzierea citirilor externe echivalente unor citiri de pe discul defect.

Deoarece o actualizare externă depune date la fel în blocul de date precum şi în corespunzătorul bloc de paritate, întârzierea totală depinde de cazul direcţării către discul defect. Rezultă trei cazuri:

- ambele scrieri locale sunt fie pe un disc valid, fie pe deja refăcuta porţiune a discului liber. Întârzierea totală este suma întârzierilor unor scrieri locale în modul normal;
- scrierea parităţii se face către discul defect. Nu este necesară paritatea scrisă, iar scrierea blocurilor de date nu necesită citire înainte de scriere. Întârzierea totală este egală cu întârzierea pentru citire a blocului de date;
- scrierea datelor către discul defect. Nu sunt necesare datele scrise, dar pagina de paritate trebuie reconstruită, necesitând citirea a *G-2* pagini.

Aproximând întârzierile citirilor locale, din nou, ca valori extreme independente a variabilelor aleatoare cu media μ şi variaţia σ^2, întârzierea medie pentru actualizare externă este $t_m = 2\mu + \left(\sqrt{\delta}\sigma / \pi\right)\ln(G-2)$

După cum s-a menţionat anterior, toate scrierile îndreptate către porţiunea discului defect, deja refăcută trebuie puse în coadă pentru a păstra integritatea blocurilor refăcute. O parte din citirile îndreptate către discul defect pot fi îndreptate către spaţiul disponibil dacă porţiunea de date a fost reconstruită. Fie procentul citirilor şi scrierilor externe către discul defect care sunt redirectate către discul disponibil la timpul t: $f_r(t)$ şi $f_w(t)$. Fie discul defect numit disc 0, iar discul liber discul C. Atunci ratele de sosire efective la timpul t ale citirilor de pe discul valid i şi de pe discul liber sunt:

$$\lambda_r(i,t) =$$

$$= \Lambda_r(1 + (G-1)(1 - f_r(t))/(C-1))/C + (G-2)(1 - f_w(t))\Lambda_w/(C(C-1))$$

$$\lambda_r(C,t) = \Lambda_r f_r(t)/C$$

Ratele de sosire efective la timpul t ale scrierilor de pe discul valid i şi de pe discul liber sunt:

$$\lambda_w(i,t) = 2\Lambda_w/C - \delta$$

pentru $0<i<C$

$$\lambda_w(C,t) = 2\Lambda_w f_w(t)/C$$

Termenul $\delta=2\Lambda_w(1-f_w(t))/(C(C-1))$ ajustează încărcările operaţiilor de citire şi scriere deoarece anumite scrieri care implică discul defect prezintă o încărcare similară cu cea a citirilor. Folosind aceste rate de sosire externe, întârzierile instantanee pentru fiecare disc şi rezultatul maxim instantaneu pot fi calculate. Întârzierile medii totale pentru citiri şi scrieri externe se pot găsi şi prin considerarea mediilor ponderate:

$$E\left[D_r^{(overall)}(t)\right] = \left((C-1)E[D_r(1)] + f_r(t)E[D_r(0)] + (1-f_r(t))E[D_r(C)]\right)/C$$

$$E\left[D_w^{(overall)}(t)\right] = \left((C-2)D_w(1,2) + f_w(t)D_w(1,C) + D_w(C,1) + (1-f_w(t))D_w(1,2)\right)$$

În analizele anterioare am presupus că procesul de refacere de pe discurile valide şi de pe cel liber operează independent. În realitate, acest lucru nu este adevărat; într-o situaţie stabilă, rata în care procesul de reconstrucţie al discului liber consuma blocuri trebuie să egaleze rata la care procesul de refacere ale discurilor valide (considerate împreună) produce blocuri.

Aproximând, alegem această rată ca fiind minimum dintre cele doua rate găsite. Fiecare bloc reconstruit pe discul liber consuma *G–1* blocuri produse de discurile valide. Prin urmare, rata instantanee a reconstrucţiei este, aproximativ:

$$\lambda_{rcc} \approx \min\left(1/E\left[Y_m^{(spare)}\right], (C-1)/\left((G-1)E\left[Y_m^{(surviving}\right]\right)\right)$$

Procentul de disc defect deja recuperat la timpul *t*, $f_w(t)$ depinde de rata de reconstrucţie $f_w(t) = B^{-1}\int_0 \lambda_{rcc}(t)dt$

De remarcat că la timpul *t* rata instantanee de refacere $\lambda_{rec}(t)$ depinde de $f_w(t)$.

Deşi nu se poate obţine o formă a expresiei mai clară pentru

$$f_w(0) = 0\,;\ f_w(t+\delta t) \approx f_w(t) + \lambda_{rcc}(t)\delta t / B$$

considerând drept 0 timpul de start al refacerii şi δt mic.

Recuperarea este completă când $f_w(t)=1$. Procentul de citiri de pe discul defect care sunt redirectate către discul liber la timpul *t* este $f_r(t)$; în mod evident acesta nu poate depăşi $f_w(t)$ deoarece doar acele blocuri deja refăcute pot fi citite de pe discul liber. În analizele noastre vom presupune că $f_r(t)/f_w(t)=\beta$ (constant).

3.6.3.3.7 Validarea rezultatelor

S-a construit un simulator *software* pentru a verifica soluţiile aproximative ale modelului descris. Acest simulator modelează

comportamentul discurilor în detaliu, de fapt urmărind mişcările capetelor de citire/scriere, iar timpii de servire necesari pentru fiecare cerere se bazează pe aceste date. Nici o aproximare nu a fost făcută cu privire la independenţa discurilor. Sunt testate atât performanţele în modul normal cât şi de recuperare prin forţarea repetată a căderii discurilor, urmărindu-se procesul de refacere; sunt măsuraţi timpii de refacere şi întârzierile de citire/actualizare. Aceste operaţii se repetă de un număr de ori suficient de mare. Se dă în Figura 3-25 şi Figura 3-26 o comparaţie între procentul datelor recuperate şi întârzierilor medii pentru accesele externe la diferite momente de timp în timpul procesului de refacere.

Toate comparaţiile sunt făcute pentru grupuri de paritate de dimensiune *G=8,* încărcările corespunzând unei utilizări de 54% în mod normal, cu 80 % dintre accese fiind citiri şi redirectarea totală a citirilor *(β=1)*.

După cum se poate vedea din grafice există o asemănare mare între aproximările analitice şi rezultatele experimentale. În majoritatea cazurilor diferenţa este mai mică decât 6%. Alte comparaţii, pentru grade diferite de utilizare, arată de asemenea o potrivire.

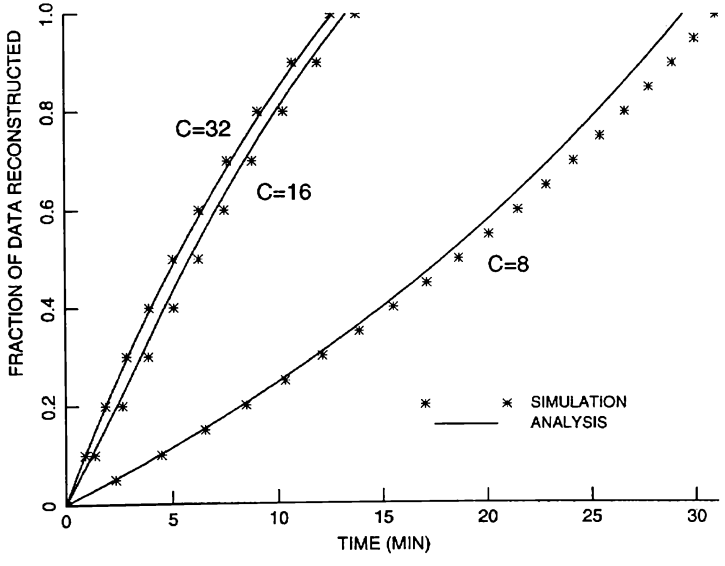

Figura 3-25

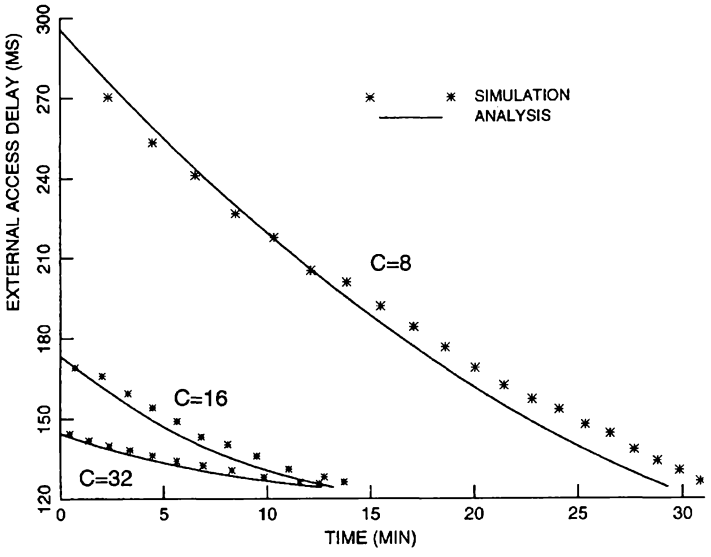

Figura 3-26

De remarcat că procesul măsurat este unul tranzitoriu şi experimentele sunt prin urmare mari consumatoare de timp. În majoritatea cazurilor unei valori generate în câteva secunde de modelul analitic îi sunt necesare câteva ore pentru a fi produsă prin simulare.

3.6.3.3.8 Evaluarea performanţelor

În acest subcapitol vom prezenta evaluarea performanţelor modelului *clustered* RAID bazat pe modelul analitic prezentat anterior. Am calculat timpii de recuperare (reconstrucţie) şi întârzierile la citire/actualizare în modul normal şi în timpul recuperării informaţiilor. Obiectul acestui studiu este acela de a înţelege:

- impactul raportului *C/G* asupra performanţelor obţinute în mod normal şi în cel al recuperării;
- efectul redirectării strategiei de citire.

După cum vom vedea pentru *unclustered* RAID, discurile „supravieţuitoare" reprezintă o gâtuire a performanţelor, iar regândirea strategiei operaţiei de citire poate îmbunătăţi performanţele. Pentru *clustered* RAID chiar şi cu un raport *C/G* mic, de valoare 2, spaţiul liber de pe disc devine o povară, iar redirectarea strategiei de citire prelungeşte timpul de reconstrucţie.

În studiul de faţă am presupus o matrice cu $G=8$ şi C variind între 8 şi 64. Ratele de sosire pentru citire/actualizare sunt considerate pentru o utilizare de 45% în modul normal şi un procent al citirilor de 0.8 (80% dintre cereri sunt citiri). În cadrul redirectării strategiei de citire, valoarea iniţială pentru raportul $\beta \equiv f_r(t)/f_w(t)$ este 1, toate cererile de citire care pot fi servite de spaţiul liber sunt aşezate în coadă. Toate valorile prezentate se bazează pe o soluţie analitică aproximativă.

Primul grafic, Figura 3-27, arată schimbarea în timpul de recuperare în momentul în care creşte raportul C/G, pentru rate de sosire a cererilor corespunzând unor utilizări de 25%, 45% şi 65% în mod normal. Pentru o utilizare de 45%, timpul de recuperare cu $C/G=2$ este jumătate din cel al *unclustered* RAID (C/G=1); la o utilizare de 65%, *unclustered* RAID se saturează — nu este posibilă o recuperare *on-line* (şi nu este punctat acest lucru în grafic). Cu $C/G \geq 2$, totuşi, este posibilă recuperarea.

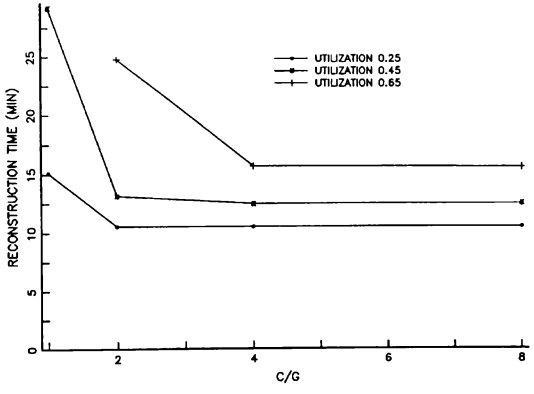

Figura 3-27

Figura 3-28 şi Figura 3-29 arată întârzierile de citire/scriere în timpul recuperării pentru $C/G=1, 2, 4$ şi 8. În ambele cazuri, întârzierile în sistemele *clustered* sunt mai scăzute decât cele din sistemele *unclustered*. Câştiguri mari se obţin prin mărirea raportului C/G de la 1 la 2; o mărire în continuare a lui C reduce destul de puţin întârzierile medii. (Această diferenţă este ridicată pentru încărcări superioare, dar principala tendinţă se păstrează — o reducere mare a întârzierilor când C/G creşte de la 1 la 2 şi avantaje mici dacă C creşte în continuare). Trebuie remarcat că procesul de reconstrucţie impune o încărcare ridicată a sistemului, după cum arată marea diferenţă dintre întârzierile normale şi întârzierile din timpul recuperării.

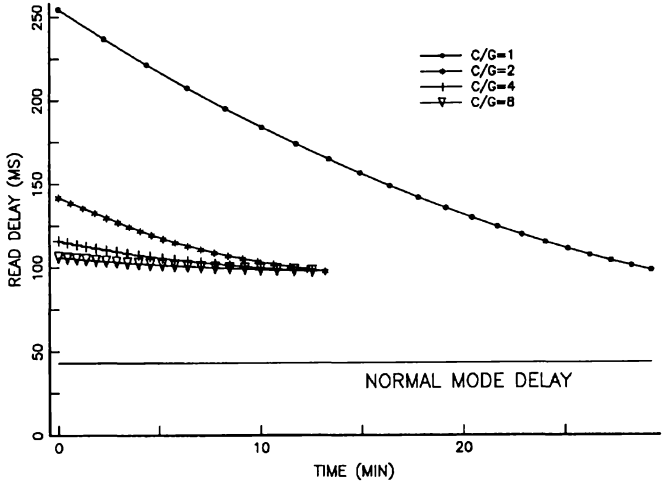

Figura 3-28

În Figura 3-30 vedem cum funcţia $f_{u}(t)$ se modifică cu timpul. O trăsătură interesantă a acestui grafic este aceea că panta curbei pentru *C/G=1* creşte cu timpul, în vreme ce panta curbei pentru *clustered arrays (C/G≥2)* descreşte cu timpul. Aceasta se explică prin faptul că în primul caz, discurile „supravieţuitoare" reprezintă o gâtuire a procesului de reconstrucţie; în vreme ce spaţiul liber este realocat, unele dintre citirile de pe discul defectat sunt transferate în spaţiul liber, iar rata de reconstrucţie a discurilor funcţionale creşte. Pentru *clustered arrays* spaţiul liber reprezintă o gâtuire a procesului de reconstrucţie şi, în vreme ce creşte $f_{u}(t)$, încărcarea spaţiului liber creşte şi ea, iar rata de reconstrucţie se află într-un uşor declin. Acest fapt explică şi comportamentul din Figura 3-31, care arată că pentru *C/G=1*, timpul de reconstrucţie scade dacă $f_{r}(t)/f_{u}(t)$ creşte şi, prin urmare, scade încărcarea discurilor nedefectate; timpul de reconstrucţie pentru *C/G≥2* arată un comportament opus. Prin urmare, strategia de redirectare a citirilor este folositoare pentru *unclustered* RAID, dar nu şi pentru *clustered* RAID, chiar şi pentru un raport scăzut al *C/G* de valoare 2. Cum citirile aleatoare deranjează procesul normal de reconstrucţie pistă cu pistă de pe discul liber în care apar şi întârzierile căutărilor în plus, strategia este valabilă doar când discurile nedefectate reprezintă o gâtuire a performanţelor în cazul *unclustered* RAID.

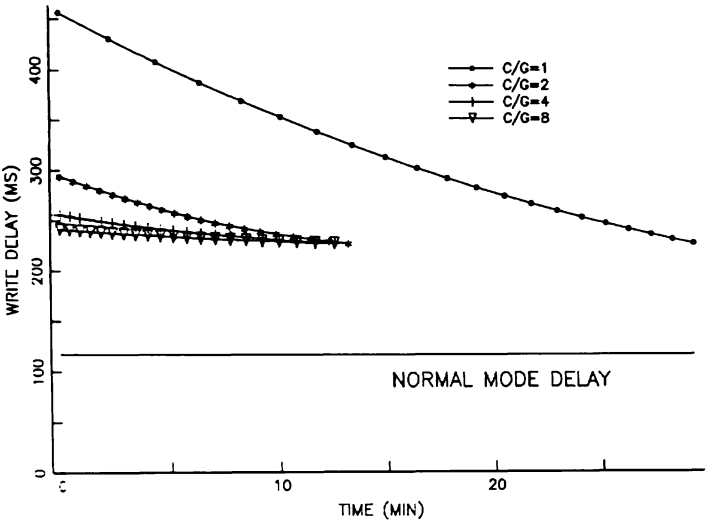

Figura 3-29

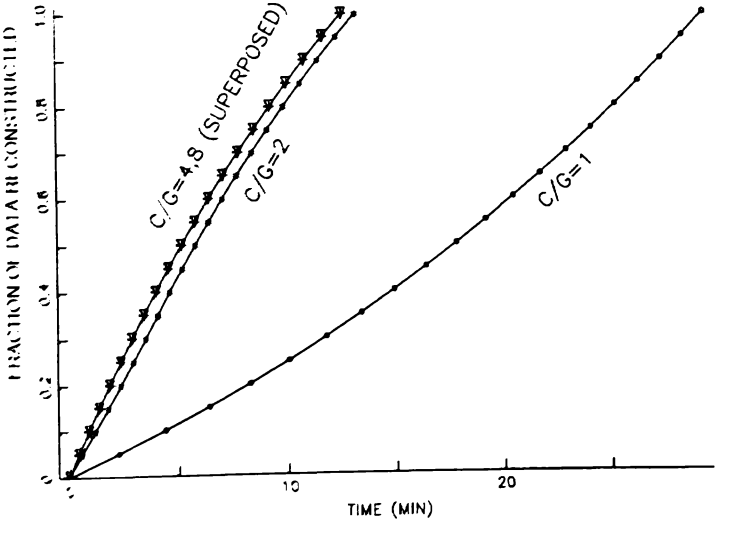

Figura 3-30

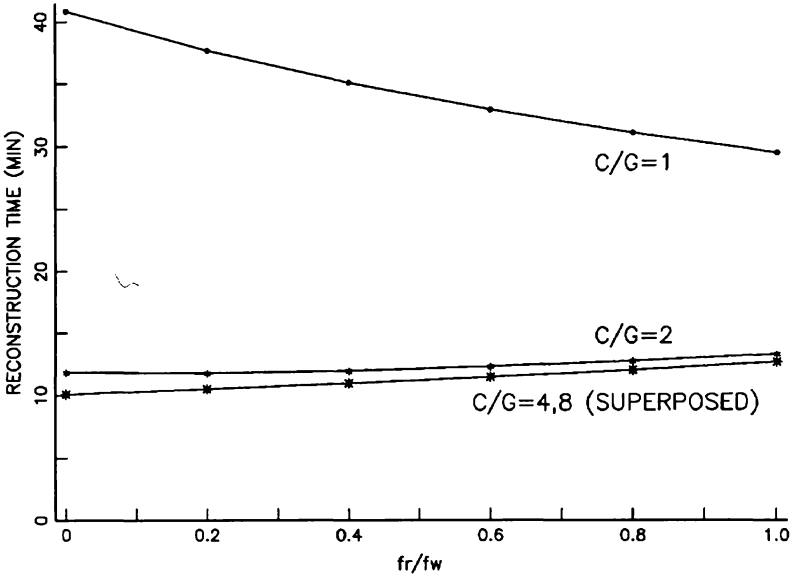

Figura 3-31

Sistemul *unclustered* este, de asemenea, sensibil şi la raportul operaţiilor de citire (procentul de cereri care sunt citiri), după cum arată Figura 3-32. În timpul recuperării, o citire de pe discul defectat care trebuie reconstruit implică *G–1* citiri de pe discul funcţional; scrierea pe acest disc implică doar *G –2* citiri de pe discurile funcţionale. Prin urmare, mărind procentul citirilor se măreşte încărcarea discurilor funcţionale în timpul recuperării; aria de discuri *unclustered* prezintă o creştere semnificativă a timpului de reconstrucţie o dată cu creşterea procentului de citiri. Matricele *clustered* sunt puţin afectate în acest caz.

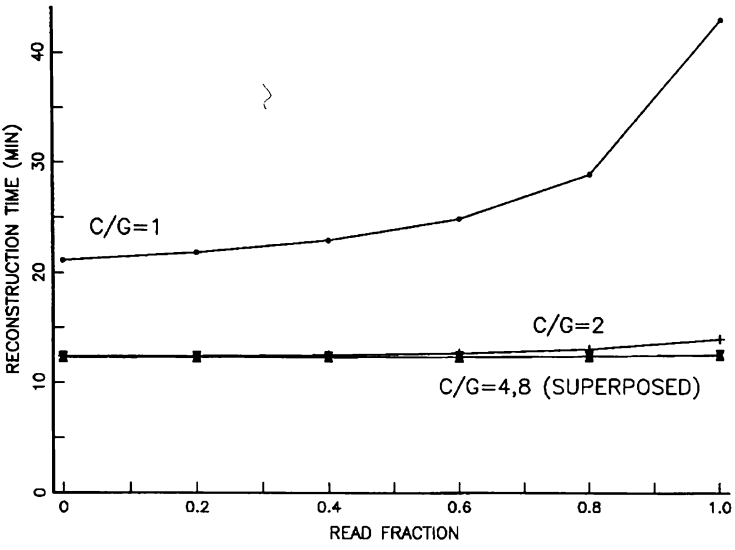

Figura 3-32

Reiese în mod clar din aceste rezultate că folosind *clustered* RAID se îmbunătăţeşte toleranţa sistemului la defectarea vreunui disc, luând în considerație fie timpul de recuperare, fie performanţele obţinute în timpul recuperării. Prin urmare, majoritatea acestor câştiguri pot fi obținute prin folosirea unor rapoarte *C/G* de valoare mică.

3.7 RAID level 6

RAID level 6, definit pentru prima dată în [Katz89] este o dezvoltare a RAID level 4 şi 5 care asigură o înaltă disponibilitate a datelor. RAID level 6 utilizează două sume de control calculate independent care protejează toate blocurile utile de date. În cazul scrierilor două blocuri de paritate independente trebuie actualizate astfel ca rezultă o puternică scădere de performanţă, ceea ce impune obligatoriu utilizarea unor metode auxiliare, cum ar fi *logging* sau *write-back cache*, pentru a reduce aceste influenţe negative.

Principalul avantaj adus de RAID level 6 constă în înalta disponibilitate a datelor. Pentru ca datele să ajungă inaccesibile datorită unor căderi de discuri, trei discuri trebuie să cadă în intervalul de timp MTTR. Această posibilitate este de mii de ori mai puţin probabilă decât cădere a două discuri într-un RAID 1 sau RAID 5.

3.7.1 Descrierea RAID level 6

După cum am mai amintit, este necesară calcularea independentă a două date de control. Aceasta se poate realiza prin utilizarea a doi algoritmi diferiţi care să acopere aceleaşi date sau utilizând aceiaşi algoritmi, dar cu acoperiri diferite ale datelor.

Utilizare a doi algoritmi diferiţi de calculare a datelor de control peste aceleaşi date este cunoscută sub denumirea de *redundanţă unidimensională* sau *P+Q parity*. În Figura 3-33 este prezentat un exemplu în care cu *Bi* am notat blocul de date utile şi cu *P-i,j*, respectiv *Q-i,j* am notat datele de control corespunzătoare blocurilor *i* şi *j*.

DISC 0	DISC 1	DISC 2	DISC 3	DISC 4	DISC 5
B0	B1	B2	B3	P-0,3	Q-0,3
B4	B5	B6	P-4,7	Q-4,7	B7
B8	B9	P-8,11	Q-8,11	B10	B11
B12	P-12,15	Q-12,15	B13	B14	B15
…	…	…	…	…	…

Figura 3-33

DISC 0	DISC 1	DISC 2	DISC 3	DISC 4	DISC 5	DISC 6	DISC 7	DISC 8	DISC 9	DISC 10
P-0,6	P-2,8	P-4,10	B0	B2	B4	P-0,2,4	B6	B8	B10	P-6,8,10
P-1,7	P-3,9	P-5,11	B1	B5	B5	P-1,3,5	B7	B9	B11	P-7,9,11
P-12,18	P-14,20	P-16,22	B12	B14	B16	P-12,14,16	B18	B20	B22	P-18,20,22
P-13,19	P-15,21	P-17,23	B13	B15	B17	P-13,16,19	B19	B21	B23	P-19,21,23
…	…	…	…	…	…	…	…	…	…	…
			DATE	DATE	DATE		DATE	DATE	DATE	

Figura 3-34

Utilizarea unui singur algoritm de calculare a datelor de control care acoperă diferite subseturi de date este cunoscută sub denumirea de *redundanţa bidimensională*. În Figura 3-34 este prezentat un exemplu în

care cu *Bi* am notat blocul *i* de date utile şi cu *P-i,j,k* am notat datele de control corespunzătoare blocurilor *i,j,k*.

Se poate observa că discurile 3, 4, 5 şi 6 formează un RAID 4, la fel ca şi discurile 7, 8, 9 şi 10. De asemenea, discurile 0, 3 şi 7, la fel ca şi 1, 4 şi 8 şi 2, 5 şi 9.

Referitor la costurile implicate de RAID level 6 cu redundanţa bidimensională se poate afirma că *MxN* discuri de date utile presupun *M+N* discuri de paritate. Pentru matrice de dimensiuni mici este scumpă din punct de vedere al capacităţii redundante (în exemplul considerat fiind de aproximativ 82%). În cazul matricelor de dimensiuni mai mari, de exemplu 100 de discuri, rezultă o redundanţă de 20%.

Deoarece funcţia *XOR* nu mai prezintă (faţă de altele) avantajul timpului de evaluare (datorită progreselor tehnologice – ex. ASICs), tendinţa în cadrul RAID 6 este către redundanţa unidimensională cu utilizarea unor coduri *Reed-Solomon*.

3.7.1.1 Comportare RAID level 6 în funcţionare normală

În funcţionare normală, procesele de scriere şi citire sunt asemănătoare cu cele întâlnite la RAID level 4 şi 5. La citire, cererile sunt direcţionate către două discuri independente obţinându-se performanţe foarte bune.

Performanţa la scriere este, însă, extrem de slabă, datorită faptului că trebuie actualizate amândouă blocurile de control implicate. Din acest motiv, utilizare unor tehnici de accelerare a scrierii, precum *logging* sau *write-back cache* se impun cu necesitate.

3.7.1.2 Protecţia la căderi în RAID level 6

Când un disc cade, fie de date sau paritate, regenerare se realizează identic cu cea în cazul RAID 5 sau 4.

O a doua cădere de disc, înainte de înlocuirea primului, va necesita pentru regenerare rezolvarea unor sisteme de două ecuaţii cu două necunoscute, întotdeauna compatibile.

3.7.1.3 Domeniile de utilizare pentru RAID level 6

Datorită preţului foarte ridicat, dar şi a fiabilităţii deosebite, această soluţie se pretează doar în acele aplicaţii care necesită asigurarea unei siguranţe foarte bune a datelor, cu acceptare unei performanţe slabe în cazul unor operaţii intensive de scriere.

3.8 Evaluarea performanţelor matricelor de discuri – RAID

În continuare, se vor compara performanţele obţinute în cazul în care toate discurile sunt operaţionale (modul normal) pentru RAID 1 şi RAID 5. Toate aceste evaluări de performanţă sunt considerate pentru două tipuri de aplicaţii:

- aplicaţii în care cererile I/O sunt pentru cantităţi mici de date (exemplu: staţii de lucru);
- aplicaţii în care cererile I/O sunt pentru cantităţi mari de date (exemplu: supercalculatoare, procesarea de imagini).

Considerăm aria de discuri fiind formată din N discuri identice. Presupunem că rotaţiile discurilor sunt sincronizate, dar nu şi cele ale braţelor. În arhitecturile considerate, data este reprezentată printr-un bloc întreţesut de-a lungul a W discuri, unde W este lăţimea secţiunii, $W<=N$. De exemplu, un fişier este divizat logic în blocuri $f_1, f_2, \ldots, f_n$, care sunt stocate pe discuri consecutive în cadrul secţiunii (n poate fi mai mare decât lăţimea secţiunii W). Dacă o cerere accesează exact un întreg multiplu de W blocuri de date dintr-o secţiune (aliniat la marginile secţiunii), atunci se numeşte o cerere I/O *full stripe* (pe toată secţiunea). Altfel, ea se numeşte cerere I/O *partial stripe* (pe o parte a secţiunii). Principalul avantaj al întreţeserii blocurilor este acela că astfel este posibilă execuţia concomitentă a mai multor cereri I/O mici. Alte alternative sunt întreţeserile la nivel de bit sau la nivel de *byte*, exemplificate prin RAID 2 şi RAID 3. Totuşi, ambele necesită ca toate discurile din matrice să fie implicate în servirea fiecărei cereri I/O şi, astfel, cel mult o cerere I/O poate fi executată la un moment dat. Cel mai semnificativ avantaj obţinut din aceste arhitecturi este o rată de transfer ridicată. În vreme ce aceste arhitecturi sunt potrivite pentru aplicaţii în care orice cerere transferă cantităţi mari de date, ele nu sunt potrivite pentru prelucrarea informaţiilor de afaceri sau a mediului staţiilor de lucru. În acest caz, deoarece fiecare cerere transferă o cantitate scăzută de date, avantajul ratei de transfer ridicate scade. Pe de altă parte, deoarece toate discurile trebuie să deservească o singură cerere, aria de discuri nu poate suporta operaţii I/O multiple în mod concurent. Un avantaj evident al întreţeserii datelor de-a lungul secţiunilor într-o matrice de discuri este încărcarea echilibrată a fişierelor des folosite pe discuri multiple.

3.8.1.1 Analiză comparativă între RAID 1 şi RAID 5 pentru volum mare de cereri

În continuare, se studiază implicarea matricelor de discuri în aplicaţii cum ar fi procesarea informaţiilor din domeniul afacerilor, precum şi al mediului staţiilor de lucru. Se vor prezenta rezultate analitice şi experimentale pentru arhitecturi de matrice de discuri împreună cu diverse strategii de proiectare. Deoarece s-a observat în literatura de specialitate că pentru aceste tipuri de aplicaţii cererile I/O accesează în mod obişnuit o cantitate mică de date, facem presupunerea că fiecare cerere accesează un singur bloc de date (4096 *bytes*) şi că fiecare bloc distinct este în mod egal accesat. În acest caz, discurile pot servi cererile în mod independent una de cealaltă. În cele din urmă presupunem că cererile sosesc în concordanţă cu procesul de sosire *Poisson*.

Cel mai important avantaj al RAID 5 în comparaţie cu RAID 1 este preţul mai scăzut al acestuia, deoarece RAID 1 necesită un număr de discuri aproape dublu pentru a atinge aceeaşi capacitate, sau, folosind acelaşi număr de discuri, RAID 1 ajunge doar la jumătate din capacitatea RAID 5. Dat fiind costul mare plătit de RAID 1, ne interesează câştigul de performanţă atins de RAID 1 faţă de RAID 5.

În acest subcapitol vom analiza RAID 5 în conjuncţie cu o strategie AR. Apoi, folosind acelaşi model, vom compara performanţele RAID 5 cu varianta *group-rotate declustering* a RAID 1 în conjuncţie cu o strategie SQ. De aici înainte, pentru a nu apărea vreo confuzie, vom folosi RAID 1 pentru a ne referi la varianta *group-rotate declustering* în combinaţie cu SQ.

Considerăm o matrice de discuri cu N discuri. Bazat pe modelul prezentat anterior, timpul de servire al discului pentru o cerere de citire este:

$$Y_r = X + T \quad (1)$$

iar pentru o cerere de scriere este

$$Y_w = X + T + R_{\max} \quad (2)$$

unde X este suma latenţelor de căutare şi de rotaţie, T este timpul de transfer (deoarece presupunem un acces la un sigur bloc, $T=\tau$) şi R_{max} este timpul pentru o rotaţie completă. (De remarcat că după timpul de transfer la o citire T, discul trebuie să se rotească până la începutul

blocului de date şi apoi să scrie din nou. Suma timpilor de rotaţie şi de transfer la scriere este egală cu timpul pentru o rotaţie completă R_{max}).

Pentru uşurinţa studiului, se introduc următoarele notaţii în analizele noastre:

- λ: rata de sosire a cererilor către subsistemului I/O;
- $\lambda_p(i)$, $\lambda_d(i)$: ratele de sosire pentru cererile prioritare de actualizare a parităţii şi cele de prioritate scăzută a citirilor/scrierilor pe discul i, $i=1, 2, ..., N$;
- $Q_p(i)$, $Q_d(i)$: r.v.; indică timpii necesari operaţiilor efectuate asupra cozilor în urma unor cereri de prioritate ridicată, respectiv scăzută asupra discului i;
- Y_p, Y_d: r.v.; pentru timpul de servire a cererilor cu prioritate ridicată şi scăzută;

În modelul considerat, cererile I/O se presupun că sosesc în aria de discuri după algoritmul *Poisson* cu rata λ. Sunt păstrate două cozi în faţa fiecărui disc, una pentru cererile de citire/scriere a datelor (coada D), iar cealaltă pentru cererile de paritate (coada P). Când soseşte o cerere, ea este îndreptată către discul având probabilitatea p_i, i=1, 2 …, N. Intrările în coada D a discului i sunt descrise în algoritmul *Poisson* prin parametrul $\lambda_d(i) = p_i \lambda$, unde cererea poate fi de citire, având probabilitatea p_r, iar cea de scriere cu probabilitatea $p_w=1-p_r$. Intrările în coada P a discului i sunt descrise printr-o suprapunere de $N-1$ cereri de paritate. Când N este mare, el poate fi aproximat în algoritmul *Poisson* prin parametrul $\lambda_p(i)=(1-p_i)p_w\lambda/(N-1)$. Cele $N-1$ procese de generare a parităţii nu sunt independente unul faţă de celalalt. Totuşi, când N este mare, corelaţiile între procese se micşorează şi, astfel, se ajunge la concluzia că pentru $N\to\infty$ ele devin independente. De fapt, când $N=16$, rezultatele experimentale arată o mare apropiere faţă de rezultatele analitice. În sfârşit, strategia primul sosit-primul servit este folosită în cazul ambelor cozi.

În acest fel, discurile i $(i=1, ..., N)$ prin două cozi bazate pe priorităţi, în care timpul de servire a cererilor cu prioritate scăzută (citiri/scrieri în coada D) sunt obţinute din relaţiile (1) şi (2). Se defineşte θ ca fiind o variabilă asociată unei cereri şi care ia valoarea 1 dacă este o citire sau 0, în caz contrar. Atunci vom avea: $Y_p=Y_w$, şi $Y_d=\theta Y_r+(1-\theta)Y_w$, unde

$$P\{\theta = 1\} = p_r \text{ şi } P\{\theta = 0\} = p_w.$$

Primele două valori pentru Y_p şi Y_d sunt :

$$\overline{Y_p} = \overline{X} + \tau + R_{\max}$$

$$\overline{Y_p^2} = \overline{X^2} + 2(\tau + R_{\max})\overline{X} + (\tau + R_{\max})^2$$

$$\overline{Y_d} = \overline{X} + \tau + p_w R_{\max}$$

$$\overline{Y_d^2} = p_r(\overline{X^2} + 2\tau\overline{X} + \tau^2) + p_w(\overline{X^2} + 2(\tau + R_{\max})\overline{X} + (\tau + R_{\max})^2$$

unde

$$\overline{X} = \overline{S} + \overline{R},\ \overline{X^2} = \overline{S^2} + 2\overline{SR} + \overline{R^2} \quad (2)$$

Bazându-ne pe rezultatele cozilor cu priorităţi non-preemptive, întârzierile medii a cozilor *P* şi *D* corespunzătoare discurilor *i* sunt:

$$\overline{Q_p(i)} = \frac{\lambda_d(i)\overline{Y_d^2} + \lambda_p(i)\overline{Y_p^2}}{2[1 - \lambda_p(i)\overline{Y_p}]} \quad (3)$$

$$\overline{Q_d(i)} = \frac{\lambda_d(i)\overline{Y_d^2} + \lambda_p(i)\overline{Y_p^2}}{2[1 - \lambda_p(i)\overline{Y_p}][1 - \lambda_d(i)\overline{Y_d} - \lambda_p(i)\overline{Y_p}]} \quad (4)$$

Timpul mediu de răspuns la o citire de pe discul *i* este:

$$\overline{Z_r(i)} = \overline{Q_d(i)} + \overline{X} + \tau \quad (5)$$

iar timpul mediu de răspuns la o scriere este:

$$\overline{Z_w(i)} = \overline{Q_d(i)} + \frac{1}{N-1}\sum_{j=1, j\neq i}^{N}\overline{Q_p(j)} + 2\overline{X} + 2\tau + R_{\max} \quad (6)$$

În (6), primii doi termeni corespund întârzierilor medii în timp ce se aşteaptă citirea blocurilor vechi de date şi de paritate; al treilea termen corespunde latenţelor medii de căutare şi de rotaţie pentru operaţiile de citire; al patrulea termen corespunde timpilor de transfer asociaţi celor două operaţii de citire; al cincilea termen corespunde unei necesare rotaţii şi scrieri a următorului bloc de paritate după citirea vechiului bloc de paritate.

În final, timpii medii de răspuns sunt calculaţi ca o medie a tuturor discurilor:

$$\overline{Z_r} = \sum_{i=1}^{N} p_i \overline{Z_r(i)}$$

$$\overline{Z_w} = \sum_{i=1}^{N} p_i \overline{Z_w(i)}$$

$$\overline{Z} = p_r \overline{Z_r} + p_w \overline{Z_w} \quad (7)$$

Rezultatele prezentate aici pentru RAID 5 sunt obţinute prin alegerea lui $p_i=1/N$, $i=1, 2, ..., N$. Deoarece în studiul de faţă RAID 5 foloseşte o unitate mică a secţiunii (4K *bytes*) şi păstrează datele pe discuri multiple, este de înţeles presupunerea făcută cum că vin cereri cu o probabilitate egală pe fiecare disc.

Compararea performanţelor este orientată în două direcţii. În primul rând, considerăm că ambele arhitecturi (RAID 1 şi RAID 5) au aceeaşi capacitate (numărul de discuri la RAID 1 este 2n, iar la RAID 5 este $n+1$), unde n a fost ales egal cu 8. În al doilea vom compara cele două arhitecturi având acelaşi număr de discuri, caz în care RAID 1 pierde jumătate din capacitatea sa. Rezultatele sunt prezentate în Figura 3-35, din care se observa că, în primul caz, RAID 1 poate suporta mai mult decât de trei ori rata de sosire a cererilor I/O decât RAID 5. În al doilea caz, RAID 1 încă poate suporta un număr de două ori mai mare de cereri I/O într-o secundă decât RAID 5. De aici deducem că RAID 1 se comportă în mod semnificativ mai bine decât RAID 5 în medii în care cererile I/O sunt scăzute, dar cu costul sporit prin dublarea numărului de discuri sau de pierderea a jumătate din capacitate.

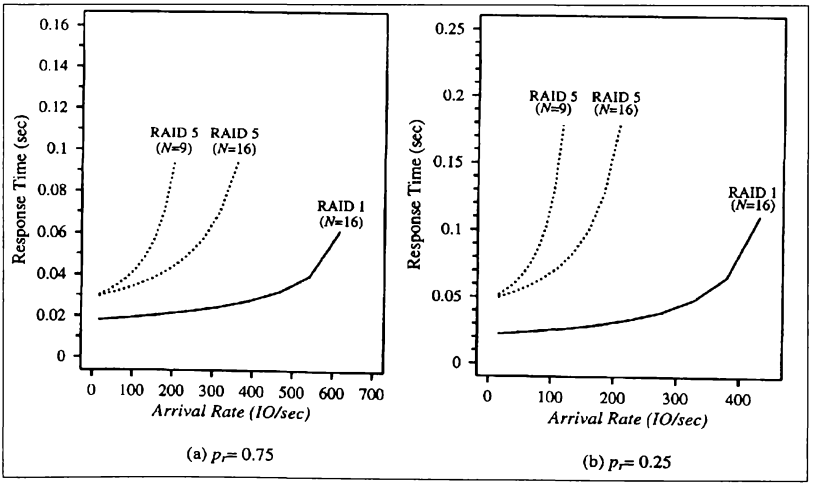

Figura 3-35

3.8.1.2 Analiză comparativă între RAID 1 şi RAID 5 pentru cereri de volum mare

Aplicaţii precum calcule superioare *(supercomputing)* şi procesare a imaginilor necesită o programare la un nivel mai scăzut şi, prin urmare, sunt necesare mai puţine cereri I/O per unitatea de timp. Totuşi, fiecare cerere I/O accesează de obicei o cantitate ridicată de date. În general, parametrii de calcul sunt mutaţi în cantitate mare de pe disc în structurile de date rezidente în memorie, iar rezultatele sunt scrise periodic înapoi pe disc. În acest caz mai multe discuri lucrează împreună ca un dispozitiv logic unic furnizând o rată de transfer mai ridicată. Cum s-a arătat anterior, se presupune că rotaţiile tuturor discurilor din matrice sunt sincronizate. Datele sunt depuse în secţiuni de-a lungul tuturor discurilor în acelaşi cilindru. Prin urmare, deoarece fiecare cerere I/O accesează una sau mai multe secţiuni, timpii de căutare se presupun a fi identici pentru fiecare disc. Uneori poate exista o diferenţă de o pistă între discuri, dar acest lucru este neglijabil. În cazul arhitecturilor *mirrored declustering* şi *group-rotate declustering*, discurile sunt împărţite în două secţiuni, unul pentru fiecare copie a datelor. Căutările se consideră că sunt sincronizate într-un grup, dar cele două grupuri pot rezolva în mod independent cereri şi pot fi modelate ca servere cu două cozi de cereri separate. Deoarece toate discurile din matrice pot fi implicate în servirea unei cereri în cazul *chained declustering* şi RAID 5, se presupune că toate sunt sincronizate şi sunt modelate ca un singur server cu o singură coadă de cereri.

În cele ce urmează vom prezenta, pe scurt, un model analitic pentru RAID 5 sincronizat şi îl vom compara cu RAID 1 pentru cazurile în care cele doua matrice au aceeaşi capacitate şi acelaşi număr de discuri, la fel cum am făcut în contextul operaţiilor reduse de I/O.

Presupunând că toate discurile din matrice sunt sincronizate, timpul de servire pentru o cerere de citire este $Y_r=X+T$. Pentru cereri de scriere, deoarece operaţii mari de I/O pot implica mai multe secţiuni de date, se disting două cazuri. În cazul *full stripe write*, în care blocurile de date care trebuie actualizate încep şi se sfârşesc la limitele secţiunii, timpul de servire a scrierilor este acelaşi cu al citirilor deoarece blocul de paritate al fiecărei secţiuni poate fi calculat din noile blocuri de date. În acest caz nu sunt necesare alte operaţii de citire în plus. Totuşi, dacă blocul de început sau de sfârşit nu este aliniat la marginile secţiunii, apare cazul parţial *stripe write* şi trebuie urmată o procedură *read-update*. Pentru simplitate, presupunem că scrierea unei secţiuni parţiale este necesară doar pentru ultima secţiune a fiecărei operaţii I/O,

considerând că blocul de început este întotdeauna aliniat la marginea sectiunii. Aceste fapte pot conduce la rezultate bune ale RAID 5.

Fie p_f probabilitatea că o cerere de scriere este o scriere în toată secţiunea. Pentru o scriere parţială în secţiune, deoarece accesează mai multe blocuri de pe fiecare disc, după citirea unui bloc din ultima secţiune (pentru calculul noului bloc de paritate), discul trebuie să se rotească la începutul celor N_s blocuri de date. Prin urmare, în locul unei rotaţii complete, este necesar ca rotaţia să fie făcută doar pe distanţa rămasă până la scrierea noilor date. Să reamintim că N_b arată numărul de blocuri de pe fiecare pistă. Timpul de rotaţie este:

$$R = \left(1 - \frac{N_s \bmod N_b}{N_b}\right) R_{\max}$$

cu valorile momentane:

$$\overline{R} = (1-\eta) R_{\max}$$

$$\overline{R^2} = \left(1 - 2\eta + \frac{1}{N_b^2} E[(N_s \bmod N_b)^2]\right) R_{\max}^2$$

unde:

$$\eta = E\left[\frac{N_s \bmod N_b}{N_b}\right] =$$

$$= \frac{1}{N} \sum_{k=0}^{N_b-1} k \sum_{m=0}^{\infty} P\{N_s = mN_b + k\}$$

şi

$$E[(N_s \bmod N_b)^2] = \sum_{k=0}^{N_b-1} k^2 \sum_{m=0}^{\infty} P\{N_s = mN_b + k\}$$

Atunci, timpul de servire pentru o scriere parţială în secţiune este:

$$Y_w = X + \tau + R + T$$

Valorile momentane pentru timpul de servire total sunt:

$$\overline{Y} = p_r \overline{Y_r} + p_w p_f \overline{Y_r} + p_w (1 - p_f) \overline{Y_w} =$$

$$= \overline{X} + \overline{T} + p_w (1 - p_f)(\tau + \overline{R})$$

$$\overline{Y^2} = \overline{X^2} + 2\overline{X}\,\overline{T} + \overline{T^2} + p_w (1 - p_f)(\overline{R^2} + \tau^2 + 2\overline{R}\,\overline{T} + 2\overline{X}\tau + 2\overline{R}\tau + 2\overline{T}\tau)$$

Având aceste rezultate RAID 5 poate fi modelat ca o coadă *M/G/1*.

În continuare, vom compara performanţele RAID 1 şi RAID 5 în cazul unor operaţii I/O de dimensiuni mari, cu o medie de 1M *bytes*.

În primul rând, vom compara RAID 1 şi RAID 5 cu aceeaşi capacitate, RAID 1 având *2n* discuri, în vreme ce RAID 5 conţine doar *n+1* discuri (cu *n=32*). Rezultatele sunt prezentate în Figura 3-36 pentru diferite valori ale lui p_f şi se observă că RAID 1 se comportă mai bine în toate cazurile decât RAID 5. Figura 3-37 ilustrează situaţia în care cele două arhitecturi au acelaşi număr de discuri. Se observă din Figura 3-37a că RAID 1 depăşeşte RAID 5 când majoritatea cererilor sunt citiri. Acest lucru se întâmplă deoarece RAID 1 serveşte două cereri de citiri simultan, în vreme ce RAID 5 secvenţial. Deşi timpul de transfer la RAID 5 este doar jumătate din cel al RAID 1, întârzierea cauzată de servirea concurentă a mai multor operaţii de citire la RAID 1 echilibrează dezavantajul unui timp de transfer mai mare. Când majoritatea cererilor sunt scrieri, RAID 1 îşi pierde avantajul de a efectua în paralel citiri. Prin urmare RAID 5 are performanţe mai bune datorită timpului de transfer mai scăzut. După cum se arată în Figura 3-37b, performanţele RAID 5 scad rapid o dată cu scăderea probabilităţii p_f datorită timpului de servire adiţional necesar procedurii *read-update* pentru scrierea parţială a unei secţiuni. Când 30% din scrieri sunt parţiale într-o secţiune, RAID 1 este echivalent cu RAID 5 la încărcări mari; iar când jumătate din scrieri sunt parţiale RAID 1 depăşeşte RAID 5. În Figura 3-38 şi Figura 3-39 este punctată sarcina maximă suportată de RAID 1 şi RAID 5 în funcţie de dimensiunea transferului sau de numărul de discuri. În general, RAID 1 se comportă mai bine decât RAID 5. Excepţia apare când se foloseşte acelaşi număr de discuri, majoritatea cererilor sunt scrieri şi dimensiunea transferului per cerere este mare (mai mare decât 2MB). În acest caz, rezultatele RAID 5 sunt mai bune decât RAID 1.

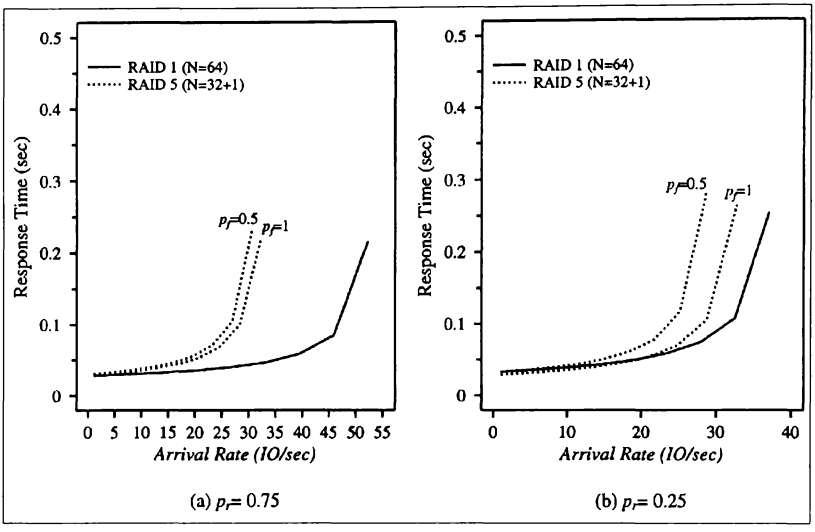

Figura 3-36

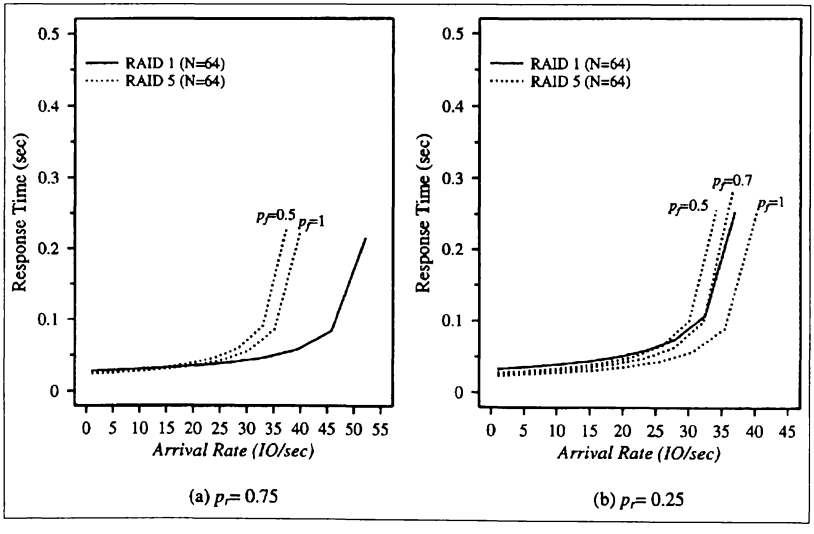

Figura 3-37

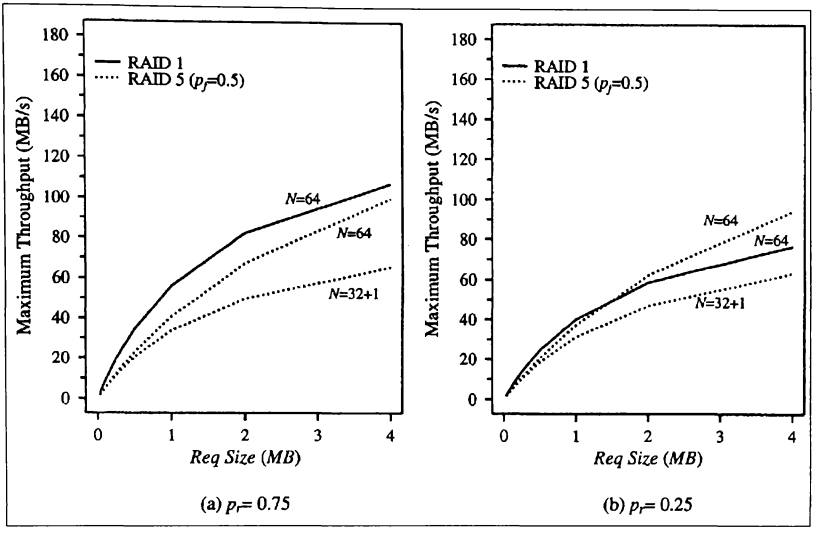

Figura 3-38

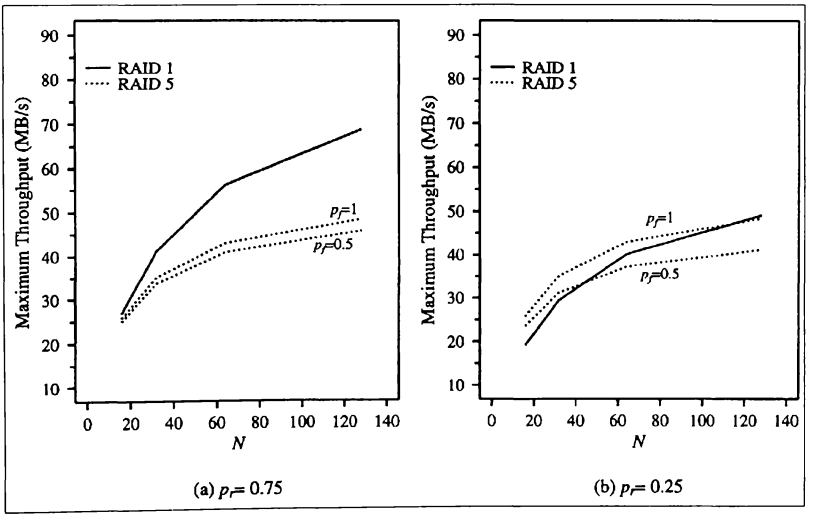

Figura 3-39

În discuţiile de mai sus am presupus ca majoritatea cererilor I/O ale RAID 5 efectuează operaţii I/O *full stripe* (p_f>0.5). Această condiţie necesită ca sistemul de fişiere sau managerul de *cache* să cunoască lăţimea secţiunii matricei de discuri. Dacă numărul de discuri creşte sau

scade, sistemul de fişiere sau managerul de *cache* trebuie să se adapteze acestei schimbări pentru a păstra un nivel ridicat al performanţelor.

3.8.1.3 Concluzii

În modul obişnuit de operare, o tactică ce repartizează o cerere de citire de pe disc unei cozi de lungime minimă, pentru orice tip posibil de date, furnizează cel mai scăzut timp mediu de răspuns dintre toate variantele RAID 1 în medii în care aplicaţiile necesită cantităţi scăzute de date. În aplicaţii în care se folosesc cantităţi mari de date, politica cozii de lungime minimă trebuie corelată cu operaţii cum ar *fi mirrored declustering* sau mai recenta operaţie *group-rotate declustering* („deciorchinizare prin rotaţie de grup"), care permite ca citiri independente să se execute independent.

Mai sus amintitele arhitecturi RAID 1 depăşesc semnificativ RAID 5 când aplicaţiile generează cereri I/O pentru cantităţi scăzute de date. Acest lucru este valabil pentru cazul în care ambele arhitecturi au acelaşi număr de discuri. În cazul aplicaţiilor care generează cereri I/O pentru cantităţi mari de date, rezultatele nu sunt la fel de evidente. RAID 5 prezintă performanţe mai ridicate când majoritatea cererilor sunt pentru cantităţi foarte mari de date, când majoritatea cererilor sunt scrieri şi când majoritatea scrierilor se efectuează pe o secţiune întreagă.

Studiile s-au efectuat prin combinarea de experimente şi analize. De exemplu, arhitectura RAID 5 a fost analizată aproximativ, dar corect, prin descompunere, în care fiecare disc este modelat după sistemul de cozi bazate pe priorităţi.

Într-o lucrare recentă, Lee şi Katz prezintă un model analitic al matricelor de discuri. În aceste lucrări, operaţiile de citire şi scriere sunt tratate în acelaşi mod la evaluarea performanţelor sistemelor RAID 5. Prin urmare, costul ridicat obţinut prin scrieri parţiale în secţiuni sub RAID 5 este ignorat.

Performanţele I/O pot fi, de asemenea, îmbunătăţite prin introducerea unui „*cache* de disc" *(disk cache)*. Eficacitatea *cache*-ului depinde de modul acceselor I/O, precum şi de dimensiunea *cache*-ului. Pentru aplicaţii în care accesele la disc denotă un factor de localizare ridicat, *disk caching*-ul poate satisface majoritatea cererilor de citire şi reduce traficul I/O cu discul. Pentru operaţiile de transfer, maparea este mai puţin eficace deoarece cererile I/O accesează în mod aleator discul şi nu este deloc practic să se mapeze întreaga baza de date. În orice caz, *cache*-urile de disc pot fi combinate cu matricele de discuri, iar studiul de faţă rămâne valabil şi în cazul acestor sisteme.

Principalele aspecte abordate în prezentul capitol sunt studiile asupra problemelor de proiectare, dezvoltarea unor modele matematice şi examinarea performanţelor diverselor arhitecturi ale matricelor de discuri, atât pentru operaţii I/O în cantitate redusă, cât şi ridicată.

În particular, am studiat arhitecturile de matrice de discuri existente şi am propus, în următorul capitol, o nouă arhitectură, RAID level 7. Se prezintă, de asemenea, strategii de proiectare potrivite pentru diverse arhitecturi de matrice de discuri. În cele din urmă, din studiile făcute s-a observat că principalul impact al performanţelor RAID 5 este costul ridicat al *partial stripe write*. Cercetătorii au căutat diverse modalităţi de a reduce acest cost. Cel mai atractiv mod este LSFS *(Log-Structured File System)*, în care multe scrieri „mici" sunt cumulate în *cache* şi apoi se face o scriere mare completă. Un alt mod este cel numit *floating parity*, în care blocurile de paritate sunt împărţite în cilindri, fiecare conţinând o pistă de rezervă. Un nou bloc de paritate, în loc să fie scris la locul său, este scris în locul celui mai apropiat bloc nealocat următor vechiului bloc de paritate. Costul plătit este următorul: controlerul trebuie să păstreze o hartă a locaţiilor tuturor blocurilor de paritate.

3.9 Concluzii

Dintre soluţiile de creştere a fiabilităţii unităţilor de discuri, prezentate anterior, în prezentul capitol, abordez, în detaliu, matricele de discuri.

Analiza performanţelor tuturor nivelurilor acceptate de RAID, împreună cu diverse strategii de accesare a discurilor, au în vedere propunerea unei noi structuri RAID, pe parcursul următorului capitol:

În această parte, teza cuprinde următoarele contribuţii:

1. o originală analiză a performanţelor structurilor RAID 1 şi RAID 5;
2. prezentarea unor modele utile în evaluarea performanţei unor structuri RAID.

4 PROPUNERE RAID LEVEL 7

Se cunosc propuneri de utilizare a unor discuri suplimentare active, dar nefolosite, în scopul oferirii unor spaţii de reconstrucţie pentru datele stocate pe discurile căzute.

Ţinând cont de viitorul acestor soluţii, propun un nou nivel de RAID, level 7, care presupune existenţa unor spaţii de rezervă, distribuite pe toate discurile din matrice, spaţii care să fie utilizate în cazul căderii oricărui disc. În funcţie de dimensiunea acestor spatii, RAID 7 poate suporta căderi multiple succesive, fără o degradare de performanţă excesivă. Evident, soluţia nu este ieftină, dar din punct de vedere a fiabilităţii, este optimă, iar din punct de vedere al performanţei este echivalentă cu RAID 5.

RAID 7 este singurul tip de RAID care poate reveni, fără intervenţie umană, după o cădere de disc, în scurt timp, la operare în regim protejat, cu performanţe apropiate de cele anterioare căderii.

Utilizare unor discuri de rezerva în matrice de tip 1–4, nu se justifică nici din punct de vedere tehnic, dar nici economic. Din acest motiv în prezenta lucrare mă voi ocupa doar de un RAID 7 provenit din dezvoltarea unei matrice RAID 5.

Este evident faptul că cu cât timpul de reparare este mai mic cu atât creste timpul până la pierderea datelor, deoarece aria revine la o stare protejată mai repede. Un mod de a „repara“ un disc căzut este de a avea rezerve de discuri, astfel încât ele pot înlocui pe cele căzute în mod automat. Timpul de reparare este practic egal cu cel de refacere a matricei.

S-a arătat că menţinerea unei rezerve active neutilizată în matrice este ineficientă. O soluţie mai bună, numită „rezervare distribuită“, este de a lăsa spaţiu suficient pe fiecare disc din matrice astfel încât datele refăcute de pe un disc căzut pot fi stocate pe acel spaţiu. Aceasta permite discurilor să fie folosite în permanenţă, mai puţine date să se piardă la o cădere de disc şi, în timpul refacerii unui disc, scrierea datelor refăcute să se facă pe mai multe discuri din matrice. Un exemplu simplu este:

D1	D2	D3	D4	D5
A	A	A	A	S
B	B	B	S	B
C	C	S	C	C
D	S	D	D	D
S	E	E	E	E

Figura 4-1

unde fiecare domeniu notat cu **S** reprezintă domeniul de rezervă. Dacă, să zicem, cade discul 1, atunci domeniul A este refăcut pe domeniul de rezervă al discului 5, domeniul B pe discul 4 s.a.m.d. Aria rezultată, având patru discuri este, încă o matrice cu toleranţă la defecte singulare, deşi fără spaţiu de rezervă.

4.1 UPGD şi rezervarea distribuită

Dacă rezervarea distribuită se combină cu o matrice UPGD, atunci rezultă o matrice cu distribuţie uniformă a grupurilor de paritate cu rezerve distribuite (UPGD/DS) *(Uniform Parity Group Distribution with Distributed Sparing)*, ca în Figura 4-2. De notat că spaţiile goale dintre domenii sunt puse doar cu scopul unei înţelegeri mai exacte, aceste spaţii nefiind fizic pe disc. Deoarece avem o UPGD, vor fi necesare exact $\lambda=2$ accese de citire pe fiecare din cele $v-1=6$ discuri din matrice pentru a reface cele $r=6$ domenii de pe discul căzut şi domeniile refăcute trebuie să fie scrise câte unul pe fiecare disc rămas, pentru un total de exact $\lambda+1=3$ accese fiecare. Astfel, aceste UPGD/DS-uri dau o distribuţie uniformă a încărcării suplimentare la căderea unui disc, atât în funcţionarea în mod degradat cât şi în timpul refacerii datele pe spaţiul de rezervă.

D1	D2	D3	D4	D5	D6	D7	
A	A	A					Grup de paritate A
B			B	B			Grup de paritate B
C					C	C	Grup de paritate C
D	D		D				Grup de paritate C
E		E			E		Grup de paritate D
F				F		F	Grup de paritate E
	G			G	G		Grup de paritate F
		H	H	H			Grup de paritate G
	I	I				I	Grup de paritate I
			J		J	J	Grup de paritate J
	K		K		K		Grup de paritate K
		L	L			L	Grup de paritate L
	M			M		M	Grup de paritate M
		N		N	N		Grup de paritate N
□	□	□	□	□	□	□	Blocuri de rezerva

Figura 4-2

Referindu-ne la Figura 4-2, după orice cădere de disc, domeniile de pe discul căzut sunt refăcute pe spaţiul de rezervă, vor fi $v'=v-1=6$ discuri cu $r'=r+1=7$ domenii în fiecare grup. În scopul obţinerii unui BIBD, astfel încât cea de-a doua cădere să genereze din nou o încărcare echilibrată, atunci $v'*r'=42$ trebuie să fie egal cu $b'*k'=42$, ceea ce se şi întâmpla, şi $\lambda'=(r'*(k'-1))/(v'-1)=2.8$ trebuie să fie întreg, şi nu este. Astfel, orice disc ar cădea, sau oricum s-ar reface datele de pe discul căzut, rezultatul nu poate fi o matrice UPGD. O a doua cădere va cauza o încărcare suplimentară, ce nu va fi distribuită în mod egal pe discurile rămase.

4.2 UPGD/DS şi căderile multiple

În continuare, se va descrie o metodă pentru matrice de discuri, astfel încât, plecându-se de la o matrice UPGD/DS, după o succesiune de căderi ale unor discuri (fiecare urmată de refacere) să rezulte tot o secvenţă de matrice UPGD/DS, fiecare bazată pe discurile rămase valide în matrice. Scopul este de a plasa grupurile de paritate şi spaţiul liber în matrice astfel încât de fiecare dată când cade un disc, aria să fie UPGD/DS înainte de cădere, dar şi după ce datele sunt refăcute de pe discul căzut şi scrise pe spaţiul liber distribuit (sau doar o UPGD dacă nu mai este spaţiu liber). Fiind realizat acest obiectiv, sarcina suplimentară în modul degradat sau/şi refacerea domeniilor de date pe spaţiul de rezervă vor fi totdeauna uniform distribuite pe discurile ramase funcţionale.

În continuare se vor folosi notaţiile:

- *N*-numărul de discuri din matrice;
- *K*-mărimea grupului de paritate;
- *b*-numărul total de grupuri de paritate;
- *(x,y)-BIBD (Balanced Complete Block Design)* format prin enumerarea tuturor combinaţiilor de *x* luate câte *y*;
- E*(x,y)*-numărul tuturor acestor combinaţii, care este:

$$x!/((x-y)!*y!)$$

- *f*-numărul de căderi la care pot fi refăcute şi remapate datele înainte ca să se înlocuiască vreun disc;
- aria se află în starea *i* dacă datele au fost refăcute şi remapate în spaţiul liber după *i* căderi. Cea de-a $i+1$-a cădere poate sau nu să fi apărut încă: ($0<=i<=f$);
- v_i-numărul de discuri ce conţin date/paritate din starea *i*. Unul din aceste discuri poate să fi căzut, dar datele/paritatea nu au fost încă remapate. Avem: $v_0=N;\ v_{i+1}=v_i-1;\ v_i=N-i$;
- r_i-numărul de domenii de date/paritate de pe fiecare disc din starea *i*, şi are valoarea: $r_i=(b*K)/v_i$;
- s_i-numărul de domenii de rezervă din fiecare disc, când aria este în starea *i*.

4.3 O analiză a algoritmul Ng-Mattson de creare a UPGD/DS

Numărul total de grupuri de paritate într-o matrice *b* nu se schimbă pe măsură ce domeniile dintr-un disc căzut se mapează în domeniile de rezervă.

Un mod simplu de aranjare pentru matrice în scopul obţinerii unei UPGD în starea *i* este de a avea grupurile de paritate posibile de mărime *K* din v_i discuri, replicate de un număr de ori. Pentru a obţine acest lucru, trebuie ca $E(v_i, K)$ să-l dividă pe *b*. Mai mult, trecând de la o matrice cu v_i discuri la una v_i-*1*, condiţia de a menţine proprietatea de simetrie a distribuţiei uniforme este să existe v_i-*K* copii ale fiecărui grup de paritate şi să mapam câte un domeniu de pe discul căzut, ce aparţine grupului, pe fiecare din cele v_i-*K* discuri neimplicate în grupul de paritate. Aceste lucru se va demonstra mai târziu. Deci, trebuie ca v_i-*K* să dividă şi el pe *b*. Prin urmare, punctul de plecare este selectarea numărului *b*, astfel încât să fie cel mai mic multiplu comun al numerelor $(v_i\text{-}K)*E(v_i,K)$, pentru toţi $i=0..f\text{-}1$.

Paşii ce urmează şi dovada că funcţionează se bazează pe presupunerea ca pentru toţi $i=0..f\text{-}1$, înainte de cea de-a $i+1$-a cădere, aria este aranjată ca o UPGD/DS bazată pe o structură (v_i,K)-*BIBD* replicată de un număr de $(v_i\text{-}K)*Q_i$ ori, unde Q_i este un întreg. Această presupunere este adevărată pentru $i=0$, prin definiţie datorită pasului 3 de mai jos şi se va arăta că va fi adevărat, prin inducţie, şi pentru $i=1..f\text{-}1$, conform pasului 4. Algoritmul pentru crearea şi funcţionarea unei matrice UPGD/DS are următorii paşi:

- selectarea numărului $f<=N\text{-}K$, care indică numărul de căderi după care se pot reface datele pe spaţiile de rezervă. Acesta este numărul de căderi până la care este nu este necesară intervenţia umană, pentru schimbarea discului sau a discurilor;
- determinarea numărului de grupuri de paritate din matrice, ca fiind c.m.m.m.c. al numerelor $(v_i\text{-}K)*E(v_i,K)$, pentru $i=0..f\text{-}1$. Aceasta înseamnă că, pentru fiecare valoare a lui *i*, există un întreg Q_i astfel încât $b=Q_i*(v_i\text{-}K)*E(v_i,K)$
- Formăm o matrice iniţială UPGD/DS prin:
  - enumerarea tuturor combinaţiilor de *N* discuri luate câte *K* şi fiecare combinaţie formează un grup de paritate;

- replicarea fiecărui grup de paritate format mai sus de Q_0 ori să obţinem în total b grupuri de paritate, cu $r=(b*K)/N$ domenii atribuite fiecăruia din cele N discuri;
- adăugând $s_0=(b*K*f)/(N*(N-F))$ domenii de rezervă pentru rezervarea distribuită;

Când aria se afla în starea i, presupunem că este aranjată ca o UPGD/DS bazată pe o structură (v_i,K)-$BCBD$ replicată de $Q_i * (v_i-K)$ ori şi cade cel de-al $i+1$-lea disc. Aria va funcţiona în mod degradat şi se va reconstrui informaţia pe spaţiul de rezervă disponibil conform astfel:

- imediat ce a căzut al $i+1$-lea disc, există v_i-1 discuri bune. Pentru uşurinţă, vom nota discurile: $D_1, ..., D_k, ..., D_{vi-1}$, şi D_{vi}, ultimul fiind cel care s-a defectat; pentru o structură (v_i,K)-$BCBD$ grupăm conceptual toate grupurile de paritate care nu folosesc discul D_{vi} într-un set numit A_i. Vor fi exact $Q_i*(v_i-K)*E(v_i-1,K)$, adică de $Q_i*(v_i-K)$ ori numărul de moduri în care se pot selecta K discuri din $D_1, D_2, ..., D_{vi}$-similar, fie B_i mulţimea tuturor grupurilor de paritate care folosesc discul D_{vi}.Vor fi exact $r_i=(b*K)/v_i=Q_i*(v_i-K)*E(v_i-1,K-1)$ adică $Q_i*(v_i-K)$ copii ale aceluiaşi grup de paritate unic care foloseşte discul D_{vi} sau, altfel spus, de $Q_i*(v_i-K)$ ori numărul de moduri în care se pot selecta $K-1$ discuri din $D_1,...,D_{vi}$.
- pentru fiecare grup de paritate unic din B_i, se iau toate *cele* $Q_i*(v_i-K)$ copii ale sale şi se reconstruiesc Q_i domenii de pe discul D_{vi} şi se remapeaza pe unul din cele v_i-K discuri nefolosite de acest grup, apoi se reconstruiesc alte Q_i domenii de pe discul D_{vi} pe un alt disc nefolosit de acest grup de paritate s.a.m.d., până când toate cele $Q_i*(v_i-K)$ domenii de pe discul D_{vi} care aparţin copiilor grupului de paritate unicat care au fost reconstruite şi remapate pe cele v_i-K discuri nefolosite de acest grup de paritate, punând Q_i domenii pe fiecare disc. Remaparea domeniilor de pe discul căzut doar pe cele ce nu folosesc acest grup de paritate oferă toleranţă la defecte deoarece nici un disc nu va conţine două domenii din acelaşi grup de paritate.

Deoarece există un număr de $E(v_i-1,K-1)$ grupuri unice de paritate în mulţimea B_i, pasul anterior se repetă de tot atâtea ori. Numărul de domenii care sunt remapate pe oricare disc D_j este de Q_i ori numărul grupurilor unicate care conţin pe D_{vi}, dar nu conţin pe D_j, adică $Q_i*E(v_i-2,K-1)$, care se poate arăta că este totuna cu $r_i/(v_i-1)$, numărul

de domenii r_i din D_{vi} distribuit în mod egal printre cele v_i-1 discuri rămase.

Propoziţie

Când aria se afla în starea $i<f$, dacă este o UPGD/DS bazată pe o structura (v_i,K)-$BCBD$ replicat de $Q_i*(v_i-K)$ ori, atunci pasul va produce o nouă structură constând din $Q_{i+1}*(v_{i+1}-K)$ replicări ale structurii (v_i,K)-$BCBD$ cu $v_{i+1}=v_i-1$ discuri.

Demonstraţie

Pentru orice K discuri $G_1,...,G_K$, avem:

- există un singur grup de paritate unicat care conţine domenii de pe discurile $G_2,...,G_K$ şi D_{vi}, şi care nu conţine pe G_1. Acest grup unicat este replicat de $Q_i*(v_i-K)$ ori. Fie P_1 această mulţime de grupuri de paritate şi ele aparţin mulţimii B_i. Q_i din domeniile ei vor fi remapate de pe discul D_{vi} pe G_1;
- fie P_2 mulţimea de $Q_i*(v_i-K)$ grupuri de paritate din mulţimea B_i care nu folosesc discul G_2, dar folosesc discurile $G_1,G_3,...,G_K$, şi D_{vi}. Q_i din domeniile sale vor fi remapate de pe discul D_{vi} pe G_2;
- ....; şi, în final fie P_k mulţimea de $Q_i*(v_i-K)$ grupuri de paritate din B_i care nu folosesc discul G_K, dar le folosesc $G_1,...,G_{K-1}$, şi $D_{vi}.Q_i$ din domeniile sale au fost remapate de pe D_{vi} pe G_k.

Astfel, Q_i*K grupuri de paritate din setul B_i au fost remapate pentru a forma Q_i*K grupuri care folosesc discurile $G_1,...,G_K$. În plus, există $Q_i*(v_i-K)$ grupuri din mulţimea A_i care, de asemenea, folosesc aceste discuri. Deci, există un total de Q_i*K grupuri care folosesc discurile $G_1,...,G_K$ în noua matrice după remapare. Deoarece acest lucru este adevărat pentru orice K discuri, este adevărat pentru oricare dintre cele $E(v_i-1,K)$ mulţimi unice de K discuri.

Deci, rezultatul remaparii îl reprezintă cele Q_i*v_i copii ale structurii (v_i-1,K)-$BCBD$. Mai mult, deoarece avem:

$$Q_i*v_i=b/[(E(v_i,K)*(v_i-K)]=\text{(din definiţia lui Qi)}$$

$$=b/\{(v_i-K)*v_i!/[(v_i-K)!*K!]\}=b/\{(v_i-1)!/[(v_i-1-K)!*K!]\}=$$

$$=b/E(v_{i+1},K)=Q_{i+1}*(v_{i+1}-K) \text{ (tot din definiţie),}$$

noua matrice este tot una UPGD/DS bazată pe o structură *(vi+1,K)-BCBD* replicată de *Qi*(vi+1-K)* ori.

Deoarece algoritmul începe cu o matrice UPGD/DS în starea 0, demonstraţia de mai sus a arătat, prin inducţie, că în urma unui număr de până la *f* căderi succesive discuri se produce o serie de noi matrice UPGD/DS. Pentru toţi *i=0,1,...,f-2,* pasul 4 produce o matrice după cea de-a *(i+1)*-a cădere. După *f* căderi, nemaifiind spaţiul liber pe discuri, va rezulta doar o matrice UPGD. Deci, algoritmul conduce la o matrice cu o funcţionare foarte bună în mod degradat şi o performanţă bună a reconstruirii.

După cea de-a *(i+1)*-a cădere şi după ce datele au fost remapate, cantitatea de spaţiu liber rămas în fiecare disc este redus de domenii remapate pe fiecare disc, adică *ri/vi-1*. Se poate arăta că după cea de-a *f*-a cădere şi după *f* remapări, cantitatea totală de spaţiu liber folosit de fiecare disc este *s0*, adică spaţiul liber cu care începe fiecare disc.

Exemplu

Algoritmul este ilustrat în Figura 4-3 pentru o matrice UPGD/DS cu *N=5*, mărimea grupului de paritate *K=3*, şi rezervare pentru *f=2* căderi. Pentru această matrice *b=20*, c.m.m.m.c al numerelor

*(v0-K)*E(v0,K)=20* şi *(v1-K)*E(v1,K)=4.* Aceasta duce la *Q0=1* şi *Q1=5.*

În Figura 4-4, grupurile de paritate şi domeniile de rezervă sunt plasate în matrice astfel încât este formată o matrice UPGD/DS din toate combinările de *N=5* discuri luate câte *K=3: 1-2-3* (grupul A), *1-2-4* (grupul B) s.a.m.d. până ce *K=3* domenii din grupul de paritate *J* sunt atribute la al zecelea triplet *3-4-5.*

Fiecare grup de paritate format mai sus este replicat de *Q0*(v0-K)=2* ori, prima replicare fiind grupurile A, B,..., J, iar cea de-a doua fiind K, L, ..., T. Rezultă astfel 12 domenii atribuite la fiecare din cele 5 discuri, *s0=(b*K*f)/(N*(N-f))=8* domenii de rezervă care se adaugă la fiecare disc.

D1	D2	D3	D4	D5
A	A	A		
B		B	B	
C	C			C
D	D	D		
E		E		E
F			F	F
	G	G	G	
	H	H		H
	I		I	I
		J	J	J
K	K	K		
L	L		L	
M	M			M
N		N	N	
O		O		O
P			P	P
	Q	Q	Q	
	R	R		R
	S		S	S
		T	T	T
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□
□	□	□	□	□

Figura 4-3

D1	D2	D3	D4	D5
A	A	A		
B		B	B	
D		D	D	
	G	G	G	
K	K	K		
L	L		L	
N		N	N	
	Q	Q	Q	
C	C	C		C
M	M		M	M
E	E	E		E
O		O	O	O
F	F		F	F
P		P	P	P
H	H	H		H
	R	R	R	R
I	I		I	I
	S	S	S	S
J		J	J	J
	T	T	T	T
□	□	□	□	
□	□	□	□	
□	□	□	□	
□	□	□	□	
□	□	□	□	

Figura 4-4

Notăm discurile D_1, D_2, D_3, D_4, D_5 (D_5 fiind cel care se defectează).

În Figura 4-4 toate grupurile de paritate care nu folosesc discul D_5 sunt puse în mulțimea A_0.

Vor fi exact $Q_0*(v_0-K)*E(v_0-1,K)=8$, adică de $Q_0*(v_0-K)=2$ ori numărul de moduri în care se pot selecta $K=3$ discuri din $v_0-1=4$ discuri. Aceste grupuri sunt A, B, D, G şi K, L, N, Q puse în topul figurii 4b. Toate grupurile care folosesc discul D_5 sunt puse în mulţimea B_0, unde vor fi exact r_0 elemente, adică $Q_0*(v_0-K)=2$ copii ale celor $E(v_0-1,K-1)$ grupuri unice care folosesc discul D_5. Aceste grupuri sunt C, M, E, O, F, P, H, R, L, S, J şi T puse în josul figurii (grupurile C şi M reprezintă 2 copii ale aceluiaşi unicat). Pentru un unicat (de exemplu C) în mulţimea B_0 se iau toate cele $Q_0*(v_0-K)=2$ copii (C şi M) şi se reconstruiesc $Q_0=1$ domeniu de pe D_5 pe fiecare din cele $v_0-K=2$ discuri nefolosite de aceste grupuri (de exemplu se reconstruieşte domeniul C de pe D_5 pe D_3 şi domeniul M de pe D_5 pe D_4). Această procedură se repetă până ce toate domeniile de pe discul D_5 au fost remapate pe discurile $D_{1..4}$. Amplasarea rezultantă a grupurilor de paritate este o matrice UPGD/DS cu $v_1=4$, $K=3$, $b=20$, $r_1=15$ şi $s_1=5$. Ea este bazată pe o matrice cu o structură (3, 4)-BCBD replicată de $Q_1*(v_1-K)=5$ ori.

La o a doua cădere, procedura se repetă cu aceleaşi valori ale lui b şi K, dar cu $v_1=4$ şi $Q_1=5$. Amplasarea grupurilor de paritate este dată în Figura 4-5 şi este o matrice UPGD/DS cu $v_2=3$, $K=3$, $b=20$, $r_2=20$ şi $s_2=0$. Următoarea cădere va forţa sistemul să funcţioneze în mod degradat până ce unul din discurile căzute este înlocuit şi îi sunt restaurate datele.

D1	D2	D3	D4
A	A	A	
K	K	K	
C	C	C	
E	E	E	
H	H	H	
B	B	B	B
M	M	M	M
F	F	F	F
I	I	I	I
D	D	D	D
N	N	N	N
O	O	O	O
P	P	P	P
J	J	J	J
G	G	G	G
Q	Q	Q	Q
R	R	R	R
S	S	S	S
T	T	T	T

Figura 4-5

4.4 Optimizarea matricelor de discuri UPGD/DS compacte

Algoritmul descris anterior va funcţiona pentru orice triplet **(n,k,f)**

unde:

- *n* reprezintă numărul iniţial de discuri în matrice;
- *k* reprezintă dimensiunea grupului de paritate;
- *f* reprezintă numărul de căderi de disc pentru care s-a prevăzut spaţiu de rezervă pe fiecare dintre discuri.

Tabelul care urmează conţine numărul grupurilor de paritate *b* pentru diferite structuri UPGD/DS, caracterizate fiecare prin tripleta *(n,k,f)*, numărul de domenii de date r_0 atribuit fiecărui disc din matrice, precum şi numărul domeniilor de rezervă s_0 cu care a fost înzestrat fiecare disc pentru ca aria să poată susţine cele *f* căderi pentru care a fost proiectată. Menţionăm că cei trei parametrii care interesează, adică *b*, r_0 şi s_0 au fost generaţi de algoritmul prezentat anterior:

n	k	f	b	r_0	s_0
6	3	2	60	30	15
8	4	2	840	420	140
12	5	2	5544	2310	462
16	5	3	720720	225225	51975
16	10	3	240240	150150	34650

Figura 4-6

Se observă că în toate cazurile, în faza iniţială, fiecare matrice este aranjată într-o structura de tip UPGD/DS bazată pe enumerarea tuturor combinaţiilor de *n* discuri luate câte *k*, replicate de un anumit număr de ori (fiecare combinaţie de discuri astfel enumerată formând un grup de paritate):

- *E (6,3)* replicate de 3 ori;
- *E (8,4)* replicate de 12 ori;
- *E (12,5)* replicate de 7 ori;
- *E (16,6)* replicate de 165 ori;
- *E (16,10)* replicate de 30 ori.

De asemenea, domeniile de rezervă care trebuie alocate fiecărui disc din matrice este multiplicat de atâtea ori de câte ori sunt replicate cele *E(n,k)* grupuri de paritate din matrice.

Acest număr mare de grupuri de paritate cerut pentru ca algoritmul *Ng-Mattson* să funcţioneze implică o dimensiune mare a tabelei care va administra aria, deci, capacitatea memoriei *cache* din controler trebuie să fie mare, ceea ce va ridica în mod substanţial costul sistemului.

O cale de a reduce acest cost este aceea de a încerca o reducere a numărului de grupuri de paritate din matrice. Mai clar spus, având **n** discuri iniţial în matrice, iar dimensiunea grupului de paritate este egală cu **k**, vom căuta o soluţie care să implice în starea de plecare (înaintea apariţiei primului disc defect) **un număr mai mic de replici** ale tuturor combinaţiilor de **n** discuri luate câte **k**, numărul acestor combinaţii dând numărul de **modele** de grupuri de paritate din matrice. Practic, se caută minimizarea numărului **b**, ceea ce automat implică micşorarea lui r_0 şi a lui s_0.

În continuare, este prezentat un exemplu în care valoarea iniţiala a lui *b* este mai mică decât cea cerută de pasul 2 al algoritmului de la capitolul precedent.

Astfel, pentru o matrice UPGD/DS cu $n=6$ discuri, dimensiunea grupului de paritate $k=3$ şi spaţiu de rezervă pentru $f=2$ defecţiuni, valoarea *b* ce s-a ales, $b=20$, care reprezintă numărul grupurilor de paritate din matrice, este mai mic decât cel dat de condiţia:

$$(v_0-k) * E(v_0-k) = 60.$$

Totuşi, şi în acest caz se poate obţine o matrice UPGD/DS repetată la care îi vom atribui atributul de **compactă**. În subcapitolul următor, prezentăm structura matricei înaintea defectării primului disc.

După ce a avut loc defectarea primului disc (fie acesta D_6), informaţia care se găsea pe acesta este reconstruită în spaţiile de rezervă ale discurilor funcţionale rămase în matrice într-o asemenea maniera încât aria rezultată este o matrice UPGD/DS, având $v_1=5$ discuri, $b=20$, $r_1=12$ domenii de date per disc, $k=3$ şi $s_1=3$ domenii de rezervă pe fiecare dintre cele 5 discuri rămase.

După ce a avut loc defecţiunea celui de-al doilea disc (fie acesta D_5), informaţia ce se afla pe acest disc este reconstruită în spaţiul de rezervă al discurilor rămase în matrice (în număr de 4), devenind doar o matrice UPGD (nu mai există domenii de rezervă pe discuri) cu $v_2=4$ discuri, $b=20$, $r_2=15$, $k=3$.

Acest exemplu demonstrează că matrice UPGD/DS replicate mai mici decât cele construite cu algoritmul *Ng-Mattson* există, dar găsirea unei căi de a reconstrui domeniile de pe discul defect pe celelalte discuri din matrice în scopul de a produce o nouă matrice UPGD este dificilă.

Găsind un **b** mai mic, din algoritmul *Ng-Mattson* putem folosi fără nici o modificare paşii 1 şi 3. Pasul 2, care decide numărul de replici se va

modifica. În fine, pasul 4 şi cel mai important, care realizează practic remaparea domeniilor discului defect în domeniile de rezervă ale discurilor funcţionale ramase în matrice, se va adapta în aşa manieră încât după remaparea despre care am vorbit, noua matrice să-şi păstreze structura de tip UPGD/DS (sau UPGD în situaţia în care nu mai există spaţiu de rezervă pe discuri).

Este necesară observaţia că numărul de domenii de rezervă atribuite iniţial fiecărui disc din matrice, $s0$, va fi şi el corespunzător mai mic:

$s0 = (b*k*f)/(n*(n-f))$, (b fiind mai mic, $s0$ va rezulta de asemenea mai mic).

4.5 Algoritm optimizat de creere UPGD/DS

Cercetările întreprinse de autor, având ca scop găsirea unei soluţii de a micşora numărul b al grupurilor de paritate din matrice, au fost încununate de succes: am reuşit găsirea unei metode prin care, pentru cazul: $\{n=2*k,\ f=2\}$ obţinem un număr semnificativ mai mic al grupurilor de paritate.

În acest caz, dimensiunea grupului de paritate trebuie să fie jumătate din numărul iniţial de discuri $(k=n/2)$, şi sunt admise două defecţiuni succesive de discuri în matrice $(f=2)$.

Lucrurile stau în felul următor: înainte de defectarea primului disc, aria este aranjată într-o structură UPGD/DS bazată pe enumerarea tuturor combinaţiilor de $n=2*k$ discuri luate de k ori (fiecare din această combinaţie reprezentând un model unic de grup de paritate), replicate de un număr de x ori. Despre acest număr x vom vorbi şi îl vom calcula puţin mai târziu, rezumându-ne acum doar la a spune că rezultă mai mic decât acel număr $Q0*(v0-k)$, care reprezenta numărul iniţial de replici ale fiecărui model unic de grup de paritate, număr cerut de pasul 2 al algoritmului descris în cadrul capitolului anterior.

Revenind la ideea începută mai sus, după ce primul disc din matrice se defectează, din numărul total de grupuri de paritate, care este egal cu $b=x*E(n,k)$, vom avea o parte din ele care utilizează discul defect, iar cealaltă parte (restul până la numărul b) care nu utilizează discul defect:

- $x*E(n-1, k-1)$ grupuri de paritate utilizează discul defect;
- $x*E(n-1,k)$ grupuri de paritate nu utilizează discul defect.

Deoarece în cazul nostru avem $n=2*k$, obţinem $E(n-1,k-1)=$

$=E(2*k-1,k-1)=E(2*k-1,2*k-1-(k-1))=E(2*k-1,k)=E(n-1,k)$, deci:

$E(n-1,k-1) = E(n-1,k)$ | înmulţim cu x

şi rezultă în final:

$$x*E(n-1,k-1) = x*E(n-1,k) = b/2,$$

ceea ce înseamnă că ***numărul grupurilor de paritate care nu utilizează discul defect este egal cu numărul grupurilor de paritate care utilizează discul defect***, lucru foarte important, care va sta la baza algoritmului de construire a matricei UPGD/DS compacte.

Să notăm cu **A** mulţimea tuturor grupurilor de paritate care utilizează discul defect, iar cu **B** mulţimea tuturor grupurilor de paritate care nu utilizează discul defect. Pe baza demonstraţiei de mai sus, rezultă:

$$card\ A = card\ B = b/2\ .$$

Pentru ca, după remaparea domeniilor discului defect, într-o manieră uniformă, în domeniile de rezervă ale discurilor funcţionale rămase în matrice, aceasta să-şi păstreze proprietatea de distribuţie uniformă a grupurilor de paritate, este necesar ca domeniile discului defect să fie în aşa fel remapate, astfel încât să rezulte o matrice aranjată într-o structură UPGD/DS bazată pe enumerarea tuturor combinaţiilor de $n-1=2*k-1$ discuri luate de k ori, replicate de $y=2*x$ ori.

Notăm cu N numărul de discuri rămase în matrice după defectarea primului disc: $N=n-1$, iar cu p numărul de domenii ale discului defect (este vorba de primul disc ce s-a defectat în matrice) ce vor fi remapate în fiecare din cele N discuri funcţionale rămase în matrice:

$$p=(card\ A)/N=(card\ B)/N=b/(2*N)$$

Din punct de vedere matematic, problema se traduce prin aceea de a stabili o **corespondenţă bijectivă** între mulţimile **A** şi **B**, ţinând cont în stabilirea acestei corespondenţe ca, în final, exact p domenii aparţinând discului defect să fie remapate în fiecare din cele N discuri funcţionale rămase în matrice, în domeniile de rezervă pe care fiecare disc le deţine.

Observăm că, atât în mulţimea **A** cât şi în mulţimea **B**, fiecare element apare de *exact* x ori (fiecare combinaţie de $n-1$ discuri luate câte $k-1$ ori apare replicată de x ori în mulţimea **A**, iar în mod similar, fiecare combinaţie de $n-1$ discuri luate câte k ori apare replicată de asemenea de x ori în mulţimea **B**). Vom descompune în continuare mulţimea **A** în x submulţimi identice din toate punctele de vedere (atât ca număr de elemente, cât şi din punct de vedere al conţinutului): $A_1, A_2, ..., A_X$.

Fiecare submulţime $\mathbf{A_i}$, unde $i=1, 2, ..., x$ este compusă din toate grupurile de paritate care utilizează discul defect, **luate o singură dată**:

$$card\ A_i = (card\ A)/x = b/(2*x) = E\ (N,k-1),\ i = 1, 2, ..., x.$$

Într-o manieră asemănătoare, vom descompune mulţimea **B** în x submulţimi identice $B_1, B_2, ..., B_X$. Fiecare submulţime $\mathbf{B_i}$, unde $i=1, 2, ..., x$, este compusă din toate grupurile de paritate care utilizează discul defect, luate o singură dată:

$$card\ B_i = (card\ B)/x = b/(2*x) = E\ (N,k),\ i = 1, 2, ..., x.$$

Stabilind o corespondenţă biunivocă între submulţimile $\mathbf{A_1}$ şi $\mathbf{B_1}$, de exemplu, aceste două mulţimi având acelaşi cardinal, respectând condiţia ca fiecare disc funcţional rămas în matrice să primească exact $p'=p/x$ domenii aparţinând discului defect, urmând apoi să aplicăm această corespondenţă asupra celorlalte perechi de submulţimi (A_i, B_i), $i=2, 3, ..., x$ şi, în final, putem spune că am stabilit o corespondenţă bijectivă între mulţimile **A** şi **B**, ceea ce ne-am şi propus, iar numărul de domenii ale discului defect care vor fi remapate pe fiecare din cele $N=n-1$ discuri funcţionale rămase în matrice va fi: p'*x=p, adică atât cât ne-am propus.

Aşadar, problema se reduce la a stabili corespondenţa biunivocă între elementele mulţimilor $\mathbf{A_1}$ şi $\mathbf{B_1}$, având grijă să îndeplinim şi condiţia de remapare uniformă enunţată mai sus.

Pentru ca explicaţiile care urmează să fie mai „aerisite“, vom folosi următoarele notaţii:

- $N=n-1$ este numărul de discuri care rămân în matrice după defectarea primului disc (acesta putând fi oricare disc din matrice);
- $M = \{ D_1, D_2, D_3, ..., D_N \}$ reprezintă o mulţime având ca elemente discurile rămase în matrice după defectarea primului disc. Etichetarea acestor discuri s-a făcut doar conceptual. Considerăm aceasta mulţime ordonată după indici (în ordine crescătoare);

Din submulţimi ale mulţimii M se construiesc cele doua mulţimi A_1 şi B_1, astfel:

- A_1={toate submulţimile ordonate cu $k-1$ elemente din mulţimea M};
- B_1={toate submulţimile ordonate cu k elemente din mulţimea M}.

Cu aceste definiţii, avem:

$A_1=\{\{D_1,D_2,...,D_{k-2},D_{k-1}\},\{D_1,D_2,...,D_{k-2},D_k\},...,D_i,D_{i+1},...,D_{k+i-3},D_{k+i-2}\},\{D_i,D_{i+1},...,D_{k+i-3},D_{k+i-1}\},...,\{D_{N-k+2},D_{N-k+3},...,D_{N-1},D_N\}\};$

$B_1=\{\{D_1,D_2,...,D_{k-1},D_k\},\{D_1,D_2,...,D_{k-1},D_{k+1}\},...,D_i,D_{i+1},...,D_{k+i-2},D_{k+i-1}\},\{D_i,D_{i+1},...,D_{k+i-2},D_{k+i}\},...,\{D_{N-k+1},D_{N-k+2},...,D_{N-1},D_N\}\}.$

Mulţimile A_1 şi B_1 au acelaşi cardinal: $card\ A_1=card\ B_1=b/(2*x)$.

A stabili o corespondenţă bijectivă între elementele mulţimilor A_1 şi B_1 înseamnă a asocia fiecărui element al mulţimii A_1 exact un element al mulţimii B_1.

Reamintim că elementele mulţimilor A_1 şi B_1 sunt la rândul lor mulţimi (pe parcursul acestui capitol, vom face deseori referire la ele numindu-le **elemente-mulţime**), mai exact, sunt submulţimi ale mulţimii M, iar un element al mulţimii A_1 are cardinalul=$k-1$, în timp ce un element al mulţimii B_1 are cardinalul=k.

Să notăm cu **l** numărul de elemente ale mulţimilor A_1 şi B_1. Cu această notaţie putem scrie cele două mulţimi astfel:

$$A_1 = \{ A_{11},A_{12},...,A_{1l} \}$$

$$B_1 = \{ B_{11},B_{12},...,B_{1l} \}$$

Numărul $p'=l/N$ ne indică $1/x$ din numărul de domenii ale discului defect reconstruite în fiecare disc rămas în matrice, în spaţiul de rezervă al acestora .

Acum, presupunând că perechea de elemente (A_{1i}, $B_{1,j}$) se află în corespondenţă, A_{1i} aparţinând mulţimii A_1 şi B_{1j} aparţinând mulţimii B_1, făcând diferenţa $B_{1j}\backslash A_{1i}=\{D_t\}$, trebuie să rezulte o mulţime formată dintr-un singur element, ceea ce reprezintă o primă restricţie în stabilirea legii de corespondenţă. Elementul D_t rezultat este un element al mulţimii M (evident).

Făcând reuniunea tuturor mulţimilor (în număr total de l) de forma:

$$\{ B_{1j}\backslash A_{1i} \},$$

unde i,j aparţin mulţimii $\{1,2,3,...,l\}$, iar A_{1i} şi B_{1j} sunt două elemente aparţinând mulţimii A_1, respectiv B_1, între care s-a stabilit corespondenţa, trebuie ca mulţimea creată ca rezultat al acestei reuniuni

să conţină toate elementele mulţimii M, fiecare fiind **replicat de *p'* ori**. Aceasta reprezintă a doua restricţie impusă (această restricţie este cel mai greu de realizat), care este chiar condiţia cerută încă din start şi dacă ea se respectă, aria de discuri rezultată după remaparea domeniilor discului defect în spaţiul de rezervă al fiecărui disc rămas în ea (remapare care se va face în acest caz **uniform**) va păstra proprietatea de UPGD/DS.

4.6 Algoritm pentru stabilirea corespondentei biunivoce între elementele mulţimilor A1 şi B1

După cum am mai spus, elementele mulăimilor A_1 şi B_1 sunt ordonate, în ordinea crescătoare a indicilor. Fiecărui element A_{1i} aparţinând mulţimii A_1 îi ataşăm mulţimea complement a elementului A_{1i} (care la rândul său este o submulţime a mulţimii M) faţă de mulţimea $M:A_{1i}<====> M \setminus A_{1i}$

Ca rezultat al acestor asocieri, alcătuim o mulţime C_1 care are tot *l* elemente, iar fiecare element din C_1 este o mulţime, în fapt fiind mulţimea complement a unui element din A_1 faţă de mulţimea M, pentru toate elementele din A_1.

Pentru situaţia noastră, în care *N=2*k-1* este identică cu mulţimea B_1, singura diferenţă fiind aceea că elementele mulţimii C_1 sunt ordonate în ordinea descrescătoare a indicilor.

Vom explica principial, în cele ce urmează, modul în care se stabileşte o corespondenţă între un element al mulţimii A_1 şi un element al mulţimii B_1:

Dintr-un element al mulţimii C_1, care la rândul ei este o mulţime de forma $C_{1z}=\{D_{zk}, D_{zk+1}, ..., D_S, ..., D_{zN}\}$ extragem elementul D_S. Această alegere a lui D_S se face după o regulă ce va fi definită mai târziu. Elementul asociat lui C_{1z} din mulţimea A_1 este de forma: $A_{1z}=\{D_{z1}, D_{z2}, ..., D_{zk-1}\}$

Adăugăm elementul D_S lui A_{1z}, rezultând:

$A_{1z}=\{D_S, D_{z1}, D_{z2}, ..., D_{zk-1}\}$,

mulţime care, după ordonarea în ordine crescătoare a indicilor, este identică cu un element din mulţimea B_1, $A_{1z}=B_{1z}$.

În acest fel, am stabilit o corespondenţă între elementul A_{1Z} aparţinând mulţimii A_1 şi elementul B_{1Z} care aparţine mulţimii B_1.

Se elimină elementul A_{1Z} din mulţimea A_1, elementul asociat C_{1Z} din mulţimea C_1 şi elementul B_{1Z} din mulţimea B_1. De asemenea, se vor căuta şi se vor elimina din mulţimile C_{1r} (care sunt elemente ale lui C_1) toate elementele D_r care, adăugate elementului A_{1r} asociat lui C_{1r}, formează, după ordonare, o mulţime identică cu B_{1Z}. Aceste suprimări vor ajuta la direcţionarea căutarilor ulterioare pentru a stabili o corespondenţă între alte două elemente din A_1, respectiv B_1, pentru ca funcţia de corespondenţă să fie bijectivă.

Revenind la aria de discuri, această corespondenţă semnifică faptul că un domeniu aparţinând discului defect a fost reconstituit, pe baza parităţii, utilizând domeniile grupului de paritate asociat elementului A_{1Z} şi a fost remapat într-unul din domeniile de rezervă ale discului numărul s (acesta fiind unul din cele N discuri funcţionale rămase în matrice. Din cele k posibilităţi în care acest domeniu putea fi remapat (însemnând cele k discuri care nu utilizează grupul de paritate din care face parte domeniul luat în discuţie) a fost aleasă, pe baza unei reguli precise, una din ele. Modul de remapare este următorul:

Primul disc rămas în matrice în care vom reconstrui un domeniu al discului defect este D_i, unde $i=1$ (rezultând aşadar discul D_1).

Se repetă de un număr de $N*p'$ ori următorii 5 paşi:

Se caută, consultând lista elementelor-mulţimi ale mulţimii C_1 începând cu primul din lista, până se găseşte primul element-mulţime în a cărui componentă se află D_i. Fie C_{iZ} acest element-mulţime. Acesta are asociat elementul-mulţime A_{iZ}, aparţinând mulţimii A_1.

Adăugând elementul D_i la A_{iZ} rezultă, după ordonarea noului element-mulţime A_{iZ}, că acesta este identic cu un element-mulţime aparţinând mulţimii B_1, notat B_{iZ}. Se consideră că s-a stabilit o corespondenţă între elementul A_{iZ} din A_1 şi elementul B_{iZ} din B_1, corespondenţă care se memorează.

Se consultă din nou lista elementelor-mulţimi ale mulţimii C_1 şi, dacă se găsesc în C_1 elemente-mulţimi care au în componenţa lor pe D_i, iar dacă prin adăugarea lui D_i la elementul-mulţime asociat, aparţinând lui A_1, se obţine, după ordonare, un element identic cu B_{iZ}, se suprimă D_i din elementul-mulţime căruia îi aparţinea înaintea executării acestui pas.

De asemenea, se elimină D_i şi din elementul-mulţime asociat, aparţinând lui A_1 (în acest element-mulţime, D_i a fost adăugat la începutul acestui pas).

Se elimina urmatoarele elemente multime: $A_{iz} \subset A_1$; $B_{iz} \subset B_1$;$C_{iz} \subset C_1$.

Se atribuie variabilei *i* valoarea *i=imodn* şi se trece la pasul 1).

În final, mulţimile A_1, B_1 şi C_1 vor fi vide.

Ca urmare a executării acestui algoritm, deoarece fiecare element D_i, *i* luând valorile de la *1* la *N* face obiectul prelucrării mulţimilor A_1, B_1 şi C_1 de *p'* ori, înseamnă că fiecare din cele *N* discuri rămase în matrice va remapa *p'* domenii ale discului defect, deci, această remapare se realizează uniform printre cele *N* discuri rămase în matrice.

Aplicând analog această lege de corespondenţă pentru toate celelalte perechi de mulţimi (A_i, B_i), *i=2,3,...,x* care sunt simple replici ale mulţimilor A_1, respectiv B_1, vom avea situaţia dorită, adică aceea că fiecare din cele *N* discuri rămase în matrice va remapa exact *p=p'*x* domenii ale discului defect. Mai mult decât atât, noua matrice va fi aranjată ca o structură UPGD/DS bazată pe enumerarea tuturor combinaţiilor de *N* discuri *(N=n-1)* luate câte *k* (fiecare din aceste combinaţii reprezentând un model unic de grup de paritate), **replicate de *2*x* ori**.

A mai rămas să explicăm cum se alege numărul *x* (numărul iniţial de replici a fiecărui model unic de grup de paritate). Acest număr este legat doar de cea de-a doua defecţiune de disc din matrice şi este ales astfel încât după reconstruirea domeniilor celui de-al doilea disc defect în spaţiul de rezervă al celor *n-2* discuri rămase în matrice, aceasta să fie aranjată într-o structură UPGD/DS.

Am reuşit să demonstrăm faptul că, alegând corespunzător numărul *x*, remaparea domeniilor în urma defecţiunii celui de-al doilea disc se poate realiza utilizând algoritmul prezentat la capitolul 4.

Pentru aceasta, ţinând cont de faptul că după prima defecţiune, avem în matrice un număr de *E(n-1,k)* modele unice de grupuri de paritate replicate de *2*x* ori, din acestea, la apariţia celui de-al doilea defect, situaţia se prezintă astfel:

- *2*x*E (n-2,k-1)* grupuri de paritate utilizează discul defect;
- *2*x*E (n-2,k)* grupuri de paritate nu utilizează discul defect.

Deoarece ne rezumăm doar la două defecte, condiţia ce se impune pentru ca aria rezultată după remapare să fie UPGD, este ca din cele *2*x*E (n-2,k-1)* grupuri de paritate ce utilizează discul defect să formăm un număr întreg de replici ale fiecărui model unic de grup de paritate ce nu utilizează discul defect. Aceasta se traduce matematic prin faptul că trebuie ales cel mai mic număr *x* pentru care *2*x* divide pe *n-2-(k-1)*, adică *2*x/(2*k-k-1)=2*x/(k-1)=q* şi *q* este cel mai mic întreg pozitiv.

Cu acest *x* calculat, după remaparea domeniilor celui de-al doilea disc defect, aria va fi aranjată într-o structură de tip UPGD bazată pe enumerarea tuturor combinaţiilor de *n-2* discuri luate câte *k*, replicate de *(2*x+q)* ori.

Exemple:

$$n = 6,\ k = 3 \Rightarrow k-1 = 2,\ x = 1,\ q = 1;$$

$$n = 8,\ k = 4 \Rightarrow k-1 = 3,\ x = 3,\ q = 2;$$

$$n = 10,\ k = 5 \Rightarrow k-1 = 4,\ x = 2,\ q = 1.$$

Referitor la numărul domeniilor de rezervă care trebuie adăugate iniţial fiecărui disc din matrice, acesta este calculat cu expresia:

$$s_0 = b*k*f/(n*(n-f)) = b*k*f/((2*k)*(2*k-f)) = b*f/(2*(2*k-f)),$$

deoarece $k = n/2$.

Algoritmul descris în acest capitol a fost verificat experimental, pentru mai multe triplete *(2*k,k,2)*, ultimul element al tripletului reprezentând numărul de defecte admise, care în descrierea noastră trebuie să fie maxim 2.

Următorul tabel este oferit spre a face o comparaţie între parametrii cei mai importanţi necesari celor 2 algoritmi:

			Algoritmul iniţial			Algoritmul îmbunătăţit		
n	k	F	b	r_0	s_0	b	r_0	s_0
6	3	2	60	30	15	20	10	5
8	4	2	840	420	140	210	105	35
10	5	2	2520	1260	315	504	252	63
12	6	2	27720	13860	2772	4620	2310	462
14	7	2	72072	36036	6006	10296	6148	858
16	8	2	720720	360360	51480	90090	45045	6435

4.7 Exemplu de aplicare a algoritmului îmbunătăţit de creare a UPGD/ DS

Algoritmul pe care tocmai l-am descris îl vom exemplifica în cele ce urmează.

Vom considera cazul în care numărul de discuri în matrice este iniţial egal cu 6, dimensiunea grupului de paritate este 3, iar spaţiul de rezervă alocat fiecărui disc pentru 2 defecţiuni (*n=6, k=3, f=2).*

Alegerea numărului iniţial de replici *x*:

Se caută cel mai mic număr întreg pozitiv *q* astfel ca *2*x/(k-1)=q.* Rezultând *x=1, q=1.*

Determinarea lui b:

$$b = x*E(n,k) = 1*E(6,3) = 20$$

Formarea matricei UPGD/DS iniţiale:

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
D	D				D
E		E	E		
F		F		F	
G		G			G
H			H	H	
I			I		I
J				J	J
	K	K	K		
	L	L		L	
	M	M			M
	N		N	N	
	O		O		O
	P			P	P
		Q	Q	Q	
		R	R		R
		S		S	S
			T	T	T
□	□	□	□	□	□
□	□	□	□	□	□
□	□	□	□	□	□
□	□	□	□	□	□
□	□	□	□	□	□

Figura 4-7

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H			H	H	
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D				D
G		G			G
I			I		I
J				J	J
	M	M			M
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-8

Enumerăm toate combinaţiile de 6 discuri luate câte 3 şi fiecare combinaţie de discuri astfel enumerată formează un grup de paritate.

Deoarece $x=1$, iniţial în matrice fiecare model de grup de paritate apare o singura dată.

Numărul de domenii asignate fiecărui disc va fi $r0=(b*k)/n=10$

Numărul de domenii de rezervă pentru fiecare disc este $s_0=b*f/((2*(2*k-f))=5$

În Figura 4-7 structura matricei UPGD/DS înaintea defectării primului disc.

Presupunem că discul etichetat cu D_6 se defectează, aria fiind formată acum din discurile D_1, D_2, D_3, D_4, D_5. În Figura 4-8, toate grupurile de paritate care nu utilizează discul defect D_6 au fost grupate conceptual (să nu se înţeleagă greşit că au fost mutate fizic), formând mulţimea B, care are $x*E(5,3)=10$ elemente (B coincide cu B_1). Elementele acestei mulţimi sunt grupate în prima parte a figurii.

Analog, toate grupurile de paritate care utilizează discul defect D_6 au fost grupate formând mulţimea A, care are $x*E(5,2)=10$ elemente (şi, aici, A coincide cu A_1). Elementele acestei mulţimi au fost grupate în continuarea mulţimii B.

Deoarece în acest caz mulţimile A şi B sunt totuna cu mulţimile A_1 şi B_1, vom aplica algoritmul folosind notaţiile A, respectiv B.

Se observă că mulţimea A este formată din toate combinaţiile de $N=5$ discuri luate câte $k-1=2$, iar mulţimea B este formată din toate combinaţiile de $N=5$ discuri, luate câte $k=3$. În ambele mulţimi, fiecare combinaţie astfel enumerată formează un grup de paritate.

După remapare, fiecare din cele $n=5$ discuri din matrice va avea: $p=p'=r0/N=2$ domenii reconstruite ale discului defect D_6, iar mulţimea A trebuie să se „transforme“ într-una de tip B, adică să fie formată din enumerarea tuturor combinaţiilor de $N=5$ discuri luate câte $k=3$ ori. Această „transformare“ se traduce de fapt prin stabilirea unei corespondenţe bijective între elementele mulţimii A şi elementele mulţimii B.

În cele ce urmează, vom parcurge ciclu cu ciclu algoritmul descris în acest capitol şi vom prezenta după fiecare ciclu starea matricei de discuri (modul cum se remapează domeniile discului defect în spaţiul de rezervă al fiecărui disc rămas în matrice.

Generarea mulţimilor A, B şi C

Figura următoare ilustrează cele trei mulţimi (structura lor în faza iniţială). Pentru a uşura discuţia, etichetăm elementele celor trei mulţimi cu A^i, B^i, respectiv C^i, unde $i=1,2,...,10$, cu observaţia că există o asociere între elementul A^i al mulţimii A şi elementul C^i al mulţimii C, pentru fiecare valoare a lui i.

Mulţimea A	Mulţimea C	Mulţimea B
$A^1=\{D_1,D_2\}$	<-asociere-> $C^1=\{D_3,D_4,D_5\}$	$B^1=\{D_1,D_2,D_3\}$
$A^2=\{D_1,D_3\}$	<-asociere-> $C^2=\{D_2,D_4,D_5\}$	$B^2=\{D_1,D_2,D_4\}$
$A^3=\{D_1,D_4\}$	<-asociere-> $C^3=\{D_2,D_3,D_5\}$	$B^3=\{D_1,D_2,D_5\}$
$A^4=\{D_1,D_5\}$	<-asociere-> $C^4=\{D_2,D_3,D_4\}$	$B^4=\{D_1,D_3,D_4\}$
$A^5=\{D_2,D_3\}$	<-asociere-> $C^5=\{D_1,D_4,D_5\}$	$B^5=\{D_1,D_3,D_5\}$
$A^6=\{D_2,D_4\}$	<-asociere-> $C^6=\{D_1,D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere-> $C^7=\{D_1,D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere-> $C^8=\{D_1,D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere-> $C^9=\{D_1,D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere-> $C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

Fiecare element al mulţimii A are ataşat elementul complementar în raport cu mulţimea $M=\{D_1,D_2,D_3,D_4,D_5\}$

Avem în total $N*p'=10$ treceri prin algoritm.

Prima trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_1. Acesta este $C^5=\{D_1,D_4,D_5\}$.

Adăugăm elementului asociat lui C^5, şi anume A^5, pe D_1. După ordonarea elementului A^5 extins, obţinem: $\{D_1,D_2,D_3\}$ care este chiar B^1. Astfel, am stabilit o corespondenţă între elementul A^5 aparţinând mulţimii A şi elementul B^1 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^5 neextins într-unul din domeniile de rezervă ale discului D_1, formându-se cu acest prilej un model de grup de paritate reprezentat de B^1.

Concret, domeniul M din D_6 se remapează într-unul din domeniile de rezervă ale discului D_1, după această remapare aria arătând ca în Figura 4-9. Se observă că acum discul D_1 are cu un spaţiu de rezervă mai puţin decât celelalte discuri funcţionale rămase în matrice. În continuare, se inspectează din nou lista elementelor mulţimii C, căutând acele elemente C^i care conţin un D_k care adăugat lui A^i, face ca după ordonarea acestuia, să fie identic cu B^1. În acest caz, elementul D_K se elimină din C^i. Aplicând această regulă pe exemplul dat, vom suprima următoarele elemente: D_3 din mulţimea C^1 şi D_2 din mulţimea C^2.

În finalul acestei treceri, vom memoram corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe: A^5 din mulţimea A, C^5 asociat lui A^5 din mulţimea C, B^1 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

Mulţimea A	Mulţimea C	Mulţimea B
$A^1=\{D_1,D_2\}$	<-asociere->$C^1=\{D_4,D_5\}$	$B^2=\{D_1,D_2,D_4\}$
$A^2=\{D_1,D_3\}$	<-asociere->$C^2=\{D_4,D_5\}$	$B^3=\{D_1,D_2,D_5\}$
$A^3=\{D_1,D_4\}$	<-asociere->$C^3=\{D_2,D_3,D_5\}$	$B^4=\{D_1,D_3,D_4\}$
$A^4=\{D_1,D_5\}$	<-asociere->$C^4=\{D_2,D_3,D_4\}$	$B^5=\{D_1,D_3,D_5\}$
$A^6=\{D_2,D_4\}$	<-asociere->$C^6=\{D_1,D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere->$C^7=\{D_1,D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere->$C^8=\{D_1,D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere->$C^9=\{D_1,D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere->$C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

A doua trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_2. Acesta este $C^3=\{D_2, D_3, D_5\}$ Adăugăm elementului asociat lui C^3 şi anume A^3, pe D_2. După ordonarea elementului A^3 extins, obţinem $\{D_1, D_2, D_4\}$ care este chiar B^2. Astfel, am stabilit o corespondenţă între elementul A^3 aparţinând mulţimii A şi elementul B^2 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^3 neextins, într-unul din domeniile de rezervă ale discului D_2, formându-se cu acest prilej un model de grup de paritate reprezentat de B^2.

Concret, domeniul I din D_6 se remapează într-unul din domeniile de rezervă ale discului D_2, după această remapare aria arătând ca în Figura 4-10. Se observă că acum discul D_2 are, de asemenea, cu un spaţiu de rezervă mai puţin. Aplicând în continuare pasul 3 al algoritmului, vom suprima următoarele elemente: D_4 din mulţimea C^1 şi D_1 din mulţimea C^6.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^3 din mulţimea A, C^3 asociat lui A^3 din mulţimea C, B^2 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

MULŢIMEA A	**MULŢIMEA C**	**MULŢIMEA B**
$A^1=\{D_1,D_2\}$	<-asociere->$C^1=\{D_5\}$	$B^3=\{D_1,D_2,D_5\}$
$A^2=\{D_1,D_3\}$	<-asociere->$C^2=\{D_4,D_5\}$	$B^4=\{D_1,D_3,D_4\}$
$A^4=\{D_1,D_5\}$	<-asociere->$C^4=\{D_2,D_3,D_4\}$	$B^5=\{D_1,D_3,D_5\}$
$A^6=\{D_2,D_4\}$	<-asociere->$C^6=\{D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere->$C^7=\{D_1,D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere->$C^8=\{D_1,D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere->$C^9=\{D_1,D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere->$C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

A treia trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutăm primul element C^i care-l conţine pe D_3. Acesta este $C^4=\{D_2, D_3, D_4\}$. Adăugăm elementului asociat lui C^4 şi anume A^4, pe D_3. După ordonarea elementului A^4 extins obţinem: $\{D_1, D_3, D_5\}$ care este chiar

B^5. Astfel, am stabilit o corespondenţă între elementul A^4 aparţinând mulţimii A şi elementul B^5 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^4 neextins, într-unul din domeniile de rezervă ale discului D_3, formându-se cu acest prilej un model de grup de paritate reprezentat de B^5.

Concret, domeniul J din D_6 se remapează într-unul din domeniile de rezervă ale discului D_3, după această remapare aria arătând ca în Figura 4-11. Discul D_3 are acum cu un spaţiu de rezervă mai puţin.

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D				D
G		G			G
I			I		I
J				J	J
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-9

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D				D
G		G			G
I		I	I		
J				J	J
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
		□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-10

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D				D
G		G			G
I		I	I		
J		J		J	
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
			□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-11

Aplicând în continuare pasul 3 al algoritmului, vom suprima următoarele elemente D_5 din mulţimea C^2, D_1 din mulţimea C^9. În finalul acestei treceri, memorăm corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe

cel corespunzător din mulţimea C. Eliminăm pe A^4 din mulţimea A, C^4 asociat lui A^4 din mulţimea C, B^5 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

Mulţimea A	Mulţimea C	Mulţimea B
$A^1=\{D_1,D_2\}$	<-asociere-> $C^1=\{D_5\}$	$B^3=\{D_1,D_2,D_5\}$
$A^2=\{D_1,D_3\}$	<-asociere-> $C^2=\{D_4\}$	$B^4=\{D_1,D_3,D_4\}$
$A^6=\{D_2,D_4\}$	<-asociere-> $C^6=\{D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere-> $C^7=\{D_1,D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere-> $C^8=\{D_1,D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere-> $C^9=\{D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere-> $C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

A patra trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_4. Acesta este $C^2=\{D_4\}$. Adăugăm elementului asociat lui C^2 şi anume A^2, pe D_4. După ordonarea elementului A^2 extins, obţinem $\{D_1,D_3,D_4\}$ care este chiar B^4. Astfel, am stabilit o corespondenţă între elementul A^2 aparţinând mulţimii A şi elementul B^4 aparţinând mulţimii B.

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D				D
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
				□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-12

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
			T	T	T
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-13

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P			P	P
		R	R		R
		S		S	S
T			T	T	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-14

Spunem că am remapat un domeniu al discului defect D6 şi anume acela care intră în componenta modelului de grup de paritate asociat lui A2 neextins, într-unul din domeniile de rezervă ale discului D4,

formându-se cu acest prilej un model de grup de paritate reprezentat de B^4. Concret, domeniul G din D_6 se remapează într-unul din domeniile de rezervă ale discului D_4, după această remapare aria arătând ca în Figura 4-12. Discul D_4 are acum cu un spaţiu de rezervă mai puţin. Aplicând în continuare pasul 3 al algoritmului, vom suprima următorul element: D_1 din mulţimea C^8. În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^2 din mulţimea A, C^2 asociat lui A^2 din mulţimea C, B^4 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

Mulţimea A	Mulţimea C	Mulţimea B
$A^1=\{D_1,D_2\}$	<-asociere->$C^1=\{D_5\}$	$B^3=\{D_1,D_2,D_5\}$
$A^6=\{D_2,D_4\}$	<-asociere->$C^6=\{D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere->$C^7=\{D_1,D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere->$C^8=\{D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere->$C^9=\{D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere->$C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

A cincea trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_5. Acesta este $C^1=\{D_5\}$. Adăugăm elementului asociat lui C^1 şi anume A^1, pe D_5. După ordonarea elementului A^1 extins, obţinem $\{D_1,D_2,D_5\}$ care este chiar B^3. Astfel, am stabilit o corespondenţă între elementul A^1 aparţinând mulţimii A şi elementul B^3 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^1 neextins, într-unul din domeniile de rezervă ale discului D_5, formându-se cu acest prilej un model de grup de paritate reprezentat de B^3.

Concret, domeniul D din D_6 se remapează într-unul din domeniile de rezervă ale discului D_5, după această remapare aria arătând ca în Figura 4-13. Discul D_5 are acum cu un spaţiu de rezervă mai puţin. Aplicând în continuare pasul 3 al algoritmului, vom suprima următorul element D_1 din mulţimea C^7. În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^1 din mulţimea A, C^1 asociat lui A^1 din mulţimea C, B^3 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

Mulţimea A	Mulţimea C	Mulţimea B
$A^6=\{D_2,D_4\}$	<-asociere->$C^6=\{D_3,D_5\}$	$B^6=\{D_1,D_4,D_5\}$
$A^7=\{D_2,D_5\}$	<-asociere->$C^7=\{D_3,D_4\}$	$B^7=\{D_2,D_3,D_4\}$
$A^8=\{D_3,D_4\}$	<-asociere->$C^8=\{D_2,D_5\}$	$B^8=\{D_2,D_3,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere->$C^9=\{D_2,D_4\}$	$B^9=\{D_2,D_4,D_5\}$
$A^{10}=\{D_4,D_5\}$	<-asociere->$C^{10}=\{D_1,D_2,D_3\}$	$B^{10}=\{D_3,D_4,D_5\}$

Deci, până acum am remapat câte un domeniu din discul defect D_6 în fiecare din discurile rămase în matrice. Începând cu următoarea trecere, vom lua iarăşi pe rând câte un disc din matrice în care vom remapa domenii din D_6.

A şasea trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_1. Acesta este $C^{10}=\{D_1, D_2, D_3\}$. Adăugăm elementului asociat lui C^{10} şi anume A^{10}, pe D_1. După ordonarea elementului A^{10} extins, obţinem $\{D_1,D_4,D_5\}$ care este chiar B^6. Astfel, am stabilit o corespondenţă între elementul A^{10} aparţinând mulţimii A şi elementul B^6 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^{10} neextins, într-unul din domeniile de rezervă ale discului D_1, formându-se cu acest prilej un model de grup de paritate reprezentat de B^6.

Concret, domeniul T din D_6 se remapează într-unul din domeniile de rezervă ale discului D_1, după această remapare aria arătând ca în Figura 4-14. Aplicând în continuare pasul 3 al algoritmului, nu vom suprima nimic.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^{10} din mulţimea A, C^{10} asociat lui A^{10} din mulţimea C, B^6 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

MULŢIMEA A	MULŢIMEA C	MULŢIMEA B
$A^6=\{D_2,D_4\}$	<-asociere->$C^6=\{D_3,D_5\}$	$B^7=\{D_2,D_3,D_4\}$
$A^7=\{D_2,D_5\}$	<-asociere->$C^7=\{D_3,D_4\}$	$B^8=\{D_2,D_3,D_5\}$
$A^8=\{D_3,D_4\}$	<-asociere->$C^8=\{D_2,D_5\}$	$B^9=\{D_2,D_4,D_5\}$
$A^9=\{D_3,D_5\}$	<-asociere->$C^9=\{D_2,D_4\}$	$B^{10}=\{D_3,D_4,D_5\}$

A şaptea trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_2. Acesta este $C^8=\{D_2, D_5\}$. Adăugăm elementului asociat lui C^8 şi anume A^8, pe D_2. După ordonarea elementului A^8 extins obţinem $\{D_2, D_3, D_4\}$ care este chiar B^7. Astfel, am stabilit o corespondenţă între elementul A^8 aparţinând mulţimii A şi elementul B^7 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^8 neextins, într-unul din domeniile de rezervă ale discului D_2, formându-se cu acest prilej un model de grup de paritate reprezentat de B^7.

Concret, domeniul R din D_6 se remapează într-unul din domeniile de rezervă ale discului D_2, după această remapare aria arătând ca în Figura 4-15. Aplicând în continuare pasul 3 al algoritmului, nu vom suprima nimic.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa,

precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^8 din mulţimea A, C^8 asociat lui A^8 din mulţimea C, B^7 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

MULŢIMEA A	**MULŢIMEA C**	**MULŢIMEA B**
$A^6=\{D_2,D_4\}$ <-asociere->$C^6=\{D_5\}$		$B^8=\{D_2,D_3,D_5\}$
$A^7=\{D_2,D_5\}$ <-asociere->$C^7=\{D_3,D_4\}$		$B^9=\{D_2,D_4,D_5\}$
$A^9=\{D_3,D_5\}$ <-asociere->$C^9=\{D_2,D_4\}$		$B^{10}=\{D_3,D_4,D_5\}$

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P			P	P
	R	R	R		
		S		S	S
T			T	T	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-15

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P	P		P	
	R	R	R		
		S		S	S
T			T	T	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-16

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O		O
	P	P		P	
	R	R	R		
		S	S	S	
T			T	T	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-17

A opta trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_3. Acesta este $C^7=\{D_3, D_4\}$. Adăugăm elementului asociat lui C^7 şi anume A^7, pe D_3. După ordonarea elementului A^7 extins obţinem $\{D_2, D_3, D_5\}$ care este chiar B^8. Astfel,

am stabilit o corespondenţă între elementul A^7 aparţinând mulţimii A şi elementul B^8 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^7 neextins, într-unul din domeniile de rezervă ale discului D_3, formându-se astfel un model de grup de paritate reprezentat de B^8.

Concret, domeniul P din D_6 se remapează într-unul din domeniile de rezervă ale discului D_3, după această remapare aria arătând ca în Figura 4-16. Aplicând în continuare pasul 3 al algoritmului, vom suprima elementul D_2 din mulţimea C^9.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^7 din mulţimea A, C^7 asociat lui A^7 din mulţimea C, B^8 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

MULŢIMEA A	**MULŢIMEA C**	**MULŢIMEA B**
$A^6=\{D_2,D_4\}$ <-asociere->	$C^6=\{D_5\}$	$B^9=\{D_2,D_4,D_5\}$
$A^9=\{D_3,D_5\}$ <-asociere->	$C^9=\{D_4\}$	$B^{10}=\{D_3,D_4,D_5\}$

A noua trecere

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_4. Acesta este $C^9=\{D_4\}$. Adăugăm elementului asociat lui C^9, şi anume A^9, pe D_4. După ordonarea elementului A^9 extins, obţinem $\{D_3, D_4, D_5\}$ care este chiar B^{10}. Astfel, am stabilit o corespondenţă între elementul A^9 aparţinând mulţimii A şi elementul B^{10} aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^9 neextins, într-unul din domeniile de rezervă ale discului D_4, formându-se astfel un model de grup de paritate reprezentat de B^{10}.

Concret, domeniul S din D_6 se remapează într-unul din domeniile de rezervă ale discului D_4, după această remapare aria arătând ca în Figura 4-17.

Aplicând în continuare pasul 3 al algoritmului, nu se modifică nimic.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^9 din mulţimea A, C^9 asociat lui A^9 din mulţimea C, B^{10} din mulţimea B.

Ilustrăm noua structură a mulţimilor A, B şi C:

MULŢIMEA A **MULŢIMEA C** **MULŢIMEA B**

$A^6=\{D_2,D_4\}$ <-asociere->$C^6=\{D_5\}$ $B^9=\{D_2,D_4,D_5\}$

A zecea trecere (ultima)

Parcurgând lista elementelor mulţimii C de sus în jos, căutam primul element C^i care-l conţine pe D_5. Acesta este $C^6=\{D_5\}$. Adăugăm elementului asociat lui C^6, şi anume A^6, pe D_5. După ordonarea elementului A^6 extins, obţinem $\{D_2, D_4, D_5\}$ care este chiar B^9. Astfel, am stabilit o corespondenţă între elementul A^6 aparţinând mulţimii A şi elementul B^9 aparţinând mulţimii B.

Spunem că am remapat un domeniu al discului defect D_6 şi anume acela care intră în componenţa modelului de grup de paritate asociat lui A^6 neextins, într-unul din domeniile de rezervă ale discului D_5, formându-se cu acest prilej un model de grup de paritate reprezentat de B^9.

Concret, domeniul O din D_6 se remapează într-unul din domeniile de rezervă ale discului D_5, după această remapare aria arătând ca în Figura 4-18, ceea ce reprezintă chiar starea matricei după remaparea tuturor domeniilor discului defect (după defectarea primului disc). Aplicând în continuare pasul 3 al algoritmului, nu se modifica nimic.

În finalul acestei treceri, vom memora corespondenţa stabilită şi eliminăm elementele-mulţime între care s-a stabilit corespondenţa, precum şi pe cel corespunzător din mulţimea C. Eliminăm pe A^6 din mulţimea A, C^6 asociat lui A^6 din mulţimea C, B^9 din mulţimea B.

Noua structură a mulţimilor A, B şi C:

Mulţimea A Mulţimea C Mulţimea B

Figura 4-18

Se observă că sfârşitul algoritmului găseşte cele trei mulţimi A, C şi B fără nici un element, ceea ce înseamnă că toate domeniile discului defect D_6 au fost remapate. De asemenea, constatăm că noua matrice este aranjată tot într-o structură UPGD/DS, bazată pe enumerarea tuturor combinaţiilor de N=5 discuri luate câte k=3, replicate de 2*x=2 ori:

- grupul DDD este o replică a grupului CCC,
- grupul GGG este o replică a grupului EEE,
- grupul I I I este o replică a grupului BBB,

şi aşa mai departe,

- grupul TTT fiind o replică a grupului HHH.

Cum numărul de replici este divizibil cu N-k=2, în urma defectării celui de-al doilea disc, reconstrucţia domeniilor acestuia în spaţiul de rezervă al celorlaltor discuri rămase în matrice se face mai simplu, pentru că se îndeplinesc condiţiile cerute de algoritmul de la capitolul 4 şi, deci, se va lucra cu acest algoritm.

Starea matricei după cea de-a doua remapare se prezintă **în** Figura 4-19**.**

D1	D2	D3	D4	D5	D6
A	A	A			
B	B		B		
C	C			C	
E		E	E		
F		F		F	
H		H	H		
	K	K	K		
	L	L		L	
	N		N	N	
		Q	Q	Q	
D	D			D	
G		G	G		
I	I		I		
J		J		J	
M	M	M			
	O		O	O	
	P	P		P	
	R	R	R		
		S	S	S	
T			T	T	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	
□	□	□	□	□	

Figura 4-19

D1	D2	D3	D4
A	A	A	
B	B		B
C	C	C	
E		E	E
F	F	F	
H		H	H
	K	K	K
L	L	L	
N	N		N
Q		Q	Q
D	D		D
G		G	G
I	I		I
J		J	J
M	M	M	
	O	O	O
	P	P	P
	R	R	R
	S	S	S
T		T	T

Figura 4-20

Se observă că ea este aranjată într-o structură UPGD bazată pe enumerarea tuturor combinaţiilor de n-2=4 discuri luate câte k=3 (este doar UPGD pentru că nu mai există spaţiu de rezervă), replicate de cinci ori.

Am realizat astfel refacerea informaţiei pe cele 4 discuri rămase.

4.8 RAID tolerant la mai multe defecte

Voi prezenta, în continuare o metodă de proiectare a matricelor de discuri tolerante la mai multe defecte – RAID – MFT.

Modelele RAID prezentate anterior sunt bazate pe multiplicarea grupurilor de paritate.

Mai mult, în algoritmul prezentat anterior se impun restricţiile {n=2*k; f=2 }, unde *n* este numărul de discuri şi *f* numărul de căderi tolerate.

Matricea de discuri va exista ca sistem UPGD/DS cu N+I discuri, şi va putea suporta I căderi, fără intervenţia omului. De fiecare dată informaţia de pe discul căzut va fi reconstruită pe celelalte tot într-o matrice UPGD/DS cu excepţia ultimei căderi, când aria va fii UPGD fără spaţiu liber. Totuşi şi-n această situaţie aria noastră va fii tolerantă la defect singular, însă imediat după aceea discul va trebui înlocuit.

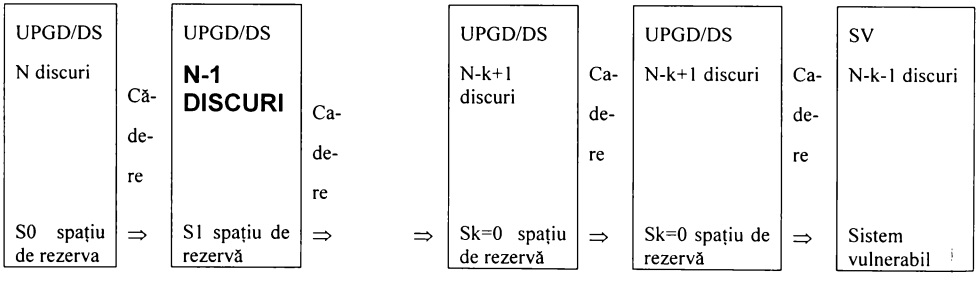

Figura 4-21

4.8.1 Algoritmul de creare UPGD/DS- MFT

Fie o UPGD/DS caracterizată prin parametrii:

- k - dimensiunea grupului de paritate;
- n - numărul de discuri;

- b - numărul de grupuri de paritate din matrice;
- c - numărul de blocuri în care se divide discul;
- r - numărul de blocuri folosite pe un disc;
- l - numărul de grupuri comune pe oricare 2 discuri;
- s - numărul de blocuri goale pe un disc (pentru DS).

Pentru a determina configuraţia matricei iniţiale, vom pleca de la ultima, cea la care va ajunge şi sistemul după *I* căderi (fără spaţiu liber), toleranţa la defect singular. Voi considera drept exemplu un RAID cu 3 discuri cu protecţia bazata pe 4 grupuri de paritate uniform distribuite,

D1	D2	D3
A	A	A
B	B	B
C	C	C
D	D	D

Figura 4-22

D1	D2	D3	D4
A		A	A
B	B		B
C	C	C	
	D	D	D

Figura 4-23

D1	D2	D3	D4
A	B	C	D
B	C	D	A
C	D	A	B

Figura 4-24

unde {A, A, A} grup de paritate etc.

Adăugând un disc în sistem şi împărţind informaţia uniform pe cele 4 discuri obţinem o matrice cu spaţiu liber, având parametrii (Figura 4-23): n=4, k=3, b=4, r = 3, l = 2

Noua configuraţie este o UPGD/DS pentru ca: r * n=b * k=12, l*(n-1)=(k-1)*r=6 şi are spaţiu liber s =1.

Am extins aria prin adăugarea unui disc.

Spaţiul liber de pe discuri se va ocupa în cazul defectării. Presupunând că discul 1 cade, reconstituirea are loc cum se arată în Figura 4-24. Nu are importanţă unde este plasat pe disc spaţiul liber. Se mai adaugă un disc.

n = 5

D1	D2	D3	D4	D5
A	B	C	D	
B	C	D	A	
C	D	A	B	

Figura 4-25

Dimensiunea grupului de paritate rămâne aceeaşi. Se va modifica însă b, numărul de grupuri. Deoarece aceste grupuri vor trebui împărţite uniform pe 5 discuri, considerăm că fiecare bloc este împărţit în 5 blocuri mai mici. Rezultă o matrice UPGD/DS cu 5 discuri şi 20 de grupuri de paritate (Figura 4-26).

D1	D2	D3	D4	D5
		A1	A1	A1
A2		A2	A2	
A3			A3	A3
A4		A4		A4
A5		A5	A5	
	B1		B1	B1
B2			B2	B2
B3	B3		B3	
B4	B4			B4
B5	B5		B5	
	C1	C1		C1
C2		C2		C2
C3	C3			C3
C4	C4	C4		
C5	C5	C5		
	D1	D1	D1	
		D2	D2	D2
	D3		D3	D3
	D4	D4		D4
	D5	D5	D5	

Figura 4-26

Se observă că s-au pus pe discul 5 blocurile cu indicele 1 de pe D1, cele cu indicele 2 de pe D2, cele cu indicele 3 de pe D3 şi cele cu 4 de pe D4.

O rearanjare duce la evidenţierea spaţiului liber, la o cădere de disc refacerea necesitând doar 3 locaţii pe fiecare disc, restul de s' = 5 rămânând disponibile pentru următoarea cădere:

D1	**D2**	**D3**	**D4**	**D5**
A2	B1	C1	D1	A1
A3	B3	C2	D2	B1
A4	B4	C4	D3	C1
A5	B5	C5	D5	B2
B2	C1	D1	A1	C2
B3	C3	D2	A2	D2
B4	C4	D4	A3	C3
B5	C5	D5	A5	D3
C2	D1	A1	B1	A3
C3	D3	A2	B2	D4
C4	D4	A4	B3	A4
C5	D5	A5	B5	B4
S'	S'	S'	S'	S'

Figura 4-27

Sistemul este UPGD/DS cu parametrii n = 5k = 3, b = 20r = 12, l = 6s = 8 (Figura 4-27).

Extindem numărul de discuri la 6. Se remarcă faptul că nu este necesară mărirea lui b (Figura 4-28).

D1	D2	D3	D4	D5	D6
		A1	A1	A1	
A2		A2	A2		
A3			A3	A3	
A4		A4		A4	
A5		A5	A5		
	B1		B1	B1	
B2			B2	B2	
B3	B3		B3		
B4	B4			B4	
B5	B5		B5		
	C1	C1		C1	
C2		C2		C2	
C3	C3			C3	
C4	C4	C4			
C5	C5	C5			
	D1	D1	D1		
		D2	D2	D2	
	D3		D3	D3	
	D4	D4		D4	
	D5	D5	D5		

Figura 4-28

Notăm altfel grupurile, de la 1 la 20 şi prezentăm sistemul extins la 6 discuri. Sistemul tolerează 3+1 căderi, ultima fiind fatală (Figura 4-29).

D1	D2	D3	D4	D5	D6
1	1		1		
2	2		2		
3	3			3	
4	4			4	
5		5		5	
6		6		6	
7		7			7
8		8			8
9			9		9
10			10		10
	11	11			11
	12	12			12
	13	13	13		
	14	14	14		
	15			15	15
	16			16	16
		17	17	17	
		18	18	18	
			19	19	19
			20	20	20

Figura 4-29

BAD	D2	D3	D4	D5	D6
	1		1		X
	2		2		X
	3		X	3	
	4		X	4	
	X	5		5	
	X	6		6	
		7		X	7
		8		X	8
		X	9		9
		X	10		10
	11	11			11
	12	12			12
	13	13	13		
	14	14	14		
	15			15	15
	16			16	16
		17	17	17	
		18	18	18	
			19	19	19
			20	20	20

Figura 4-30

BAD	D2	D3	D4	D5	BAD
	1		1	X	
	2		2	X	
	3		3	3	
	4		4	4	
	5	5		5	
	6	6		6	
	X	7		7	
	X	8		8	
	X	9	9		
		10	10	X	
	11	11	X		
	12	12	X		
	13	13	13		
	14	14	14		
	15	X		15	
	16		X	16	
		17	17	17	
		18	18	18	
		X	19	19	
		X	20	20	

Figura 4-31

BAD	BAD	D3	D4	D5	BAD
		X	1	1	
		X	2	2	
		X	3	3	
		X	4	4	
		5	X	5	
		6	X	6	
		7	X	7	
		8	X	8	
		9	9	X	
		10	10	10	
		11	11	X	
		12	12	X	
		13	13	X	
		14	14	X	
		15	X	15	
		X	16	16	
		17	17	17	
		18	18	18	
		19	19	19	
		20	20	20	

Figura 4-32

În Figura 4-30, Figura 4-31 şi Figura 4-32 se prezintă o modalitate de supravieţuire a matricei după căderea consecutivă a discurilor 1, 6 şi 2.

A căzut discul 1, apoi 6, apoi 2, refacerea căsuţelor marcate cu X făcându-se la fiecare etapă.

4.8.2 Generalizare

O matrice pentru a fi UPGD/DS trebuie să îndeplinească cerinţele următoare:

- $r * n = b * k = 12$

- $l * (n-1)=(k-1) * r=6$

Există şi cerinţele k mic şi b minim. Evident k <= n.

Numărul total de blocuri este c * n (numărul de blocuri/disc * numărul de discuri).

Fie N iniţial. Adăugăm un disc. Datele se distribuie uniform pe $n=N+1$ discuri în noua configuraţie. Se procedează la împărţirea fiecărui disc în $N+1=n$ părţi egale. Pe fiecare disc va fi informaţie $r=c * ((n-1)/n)$, iar spaţiul liber va fi $s=c * (1/n)$.

Cum $c.m.m.d.c.(n-1,n)=1 \Rightarrow n \mid c$

Extindem aria la $N+2$ discuri, distribuind informaţia. Rezultă pe fiecare disc avem $r=c * ((n-2)/n)$, spaţiul liber fiind $s=c * (2/n)$, de unde $n \mid c*2$ şi $n \mid c*(n-2)$.

Prin inducţie după i, extinzând aria la $N+i$ discuri, vom avea $c*N$ informaţie pe $N+i$ discuri, $r=c * (N/(N+i))$, $s=c * (i/(N+i))$, rezultând condiţiile: $n \mid c*(n-i)$ şi $n \mid c*i$

De unde se deduc:

- b numărul de grupuri în matrice
- r numărul de blocuri ocupate cu informaţie
- s numărul de blocuri goale

Acest sistem RAID este tolerant la $i+1$ defecte, dar nu la 2 simultane.

Suportă i defecte deoarece are spaţiu liber, iar al $i+1$-lea este tolerat datorită parităţii. Abia după aceasta este necesară intervenţia omului pentru înlocuirea unui disc defect.

Atunci când cade un disc (sistemul trece de la $N+i$ la $N+i-1$ discuri), toată informaţia de pe acel disc este împărţită egal între cele rămase. Este ca şi cum discul căzut ar fi copiat în spaţiul distribuit pe celelalte, însă el nu este copiat ci este refăcut cu ajutorul parităţii.

În tabelul de mai jos se dau câteva valori obţinute prin rularea unui program care calculează numărul de grupuri de paritate b, numărul de blocuri c pe un disc, spaţiul folosit pe discuri r, spaţiul liber s. Are ca parametrii de intrare dimensiunea grupului de paritate k, numărul de discuri după căderi şi numărul de căderi f la care poate rezista.

Ultimul algoritm prezentat dă rezultate bune când k este mic (3-5) sau apropiat de n şi se obţin valori rezonabile pentru serii de până la 5-6 căderi.

Valori de intrare			Valori obţinute						
K	N	F	f	n	b	c	r	l	s
3	3	7	1	3	1	1	1	1	0
			2	4	4	4	3	2	1
			3	5	20	20	12	6	8
			4	6	20	20	10	4	10
			5	7	140	140	60	20	80
			6	8	280	280	105	30	175
			7	9	840	840	280	70	560
3	4	5	1	4	4	3	6	3	0
			2	5	20	15	5	2	1
			3	6	20	15	30	10	12
			4	7	140	105	105	30	63
			5	8	280	210	280	70	224
3	6	5	1	6	20	10	10	4	0
			2	7	140	70	60	20	10
			3	8	280	140	105	30	35
			4	9	840	420	280	70	140
			5	10	840	420	252	56	168
3	8	4	1	8	280	105	105	30	0
			2	9	840	315	280	70	35
			3	10	840	315	252	56	63
			4	11	9240	3465	2520	504	945
4	5	4	1	5	5	4	4	3	0
			2	6	15	12	10	6	2
			3	7	105	84	60	30	24
			4	8	210	168	105	45	63
4	6	4	1	6	15	10	10	3	0
			2	7	105	70	60	6	10
			3	8	210	140	105	30	35
			4	9	630	420	280	45	140

K	N	F	f	n	b	c	r	l	s
6	6	4	1	6	1	1	1	6	0
			2	7	7	7	6	30	1
			3	8	28	28	21	45	7
			4	9	84	84	56	105	28

Figura 4-33

Valorile obţinute pentru parametrii matricelor extinse sunt foarte bune, însă nu toate soluţiile se pot aplica. De altfel, pentru unele date de intrare, nu se pot obţine soluţii, deci k şi N au anumite restricţii.

4.9 Concluzii

Principalele contribuţii aduse în acest capitol se referă la:

1. definire unei noi clase de structuri RAID (RAID 7) cu performanţe deosebite referitoare la timpii de regenerare a datelor şi reconstrucţie a matricei;
2. elaborarea a trei algoritmi performanţi pentru construcţia matricelor RAID 7.

5 RAIC – MATRICE DE CANALE REDUNDANTE ŞI INDEPENDENTE

5.1 Introducere

Principala activitate (din punct de vedere al consumului de timp) desfăşurată de un sistem de calcul o reprezintă transferul de date, astfel în acest domeniu vom regăsi principalele surse de erori precum şi posibilităţile cele mai mari de creştere a performanţelor. Chiar dacă sistemul este proiectat pentru înalta fiabilitate, el este expus la erori generate de evenimente atmosferice (umiditate, temperatură etc.), zgomot electromagnetic, căderi *hardware* şi *software*. În aceste condiţii, este foarte important ca sistemul să poate detecta erorile şi să fie capabil să execute un set de acţiuni în vederea eliminării efectelor acestora.

În continuare, vom presupune că în orice transfer de informaţii sunt implicate patru entităţi, şi anume:

- Emiţătorul sau sursa de informaţii;
- Canalul sau mediul de comunicaţii;
- Receptorul sau destinaţia datelor;
- Sursa de erori.

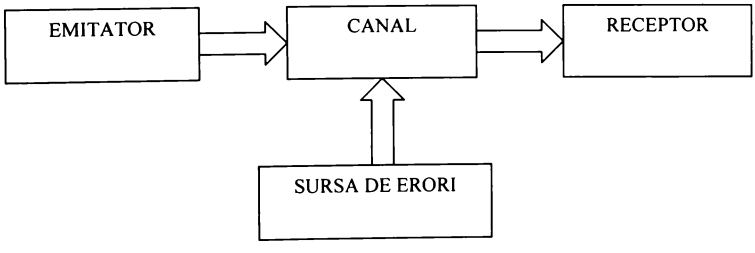

Figura 5-1

Structura ideală a suportului pentru un transfer de date

Deşi pot să apară erori la oricare din cele trei niveluri, în continuare ne vom concentra atenţia doar asupra celor referitoare la canale.

În prezenta lucrare voi considera următoarea clasificare a canalelor din punct de vedere al transformărilor logice suferite de date în timpul trecerii prin canale:

Canale cu transfer identic furnizează receptorului datele exact în aceiaşi formă ca la intrare, instantaneu (fără întârzieri).

Canalele procesor modifică datele în timpul transferului şi le furnizează instantaneu receptorului.

Canalele cu memorie furnizează receptorului datele exact în aceiaşi formă ca la intrare, dar cu o întârziere controlată.

În realitate, nu există canale care să se încadreze strict într-una din cele trei categorii. De obicei, ele sunt o combinaţie ponderată a celor trei, un exemplu fiind prezentat în Figura 5-2.

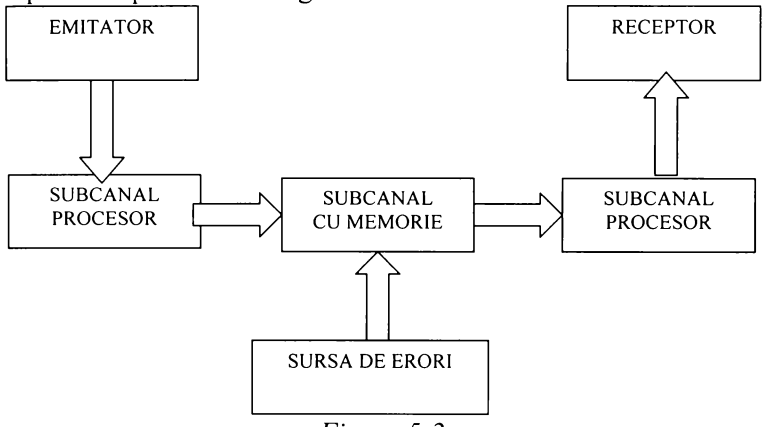

Figura 5-2

Fiecare subcanal, pe lângă funcţia sa de transfer, va prezenta un timp de propagare. De exemplu, subcanalul procesor $\Delta t_{pi,}$ iar subcanalul cu memorie $\Delta t_{m.}$ Particularizând, funcţiile de transfer şi timpii de propagare se pot practic obţine orice tip de canale.

Exemplu

Pentru un canal de tip RS 232C putem identifica:

Subcanalul procesor 1 care se ocupă de calcularea bitului de paritate şi serializarea datelor (Δt_{p1})

Subcanalul cu memorie constituit din emiţătoare, receptoare şi linia pe care se execută comunicaţia propriu-zisă, având toate împreună timpul de propagare $\Delta t_{m.}$

Subcanalul procesor 2 care restabileşte data transmisă, recalculează bitul de paritate şi verifică integritatea datei recepţionate (Δt_{pi}).

Bazat pe cele prezentate anterior, devine evident că un disc magnetic poate fi considerat un canal de comunicaţii, conectat la un receptor şi emiţător, grupate în aceiaşi unitate logică.

Generalizarea poate merge mai departe, considerând sistemele de calcul drept un set de emiţătoare, receptoare şi canale de comunicaţii, fiecare activitate fiind de fapt un transfer de date. În concluzie, problemele legate de creşterea fiabilităţii şi a performanţelor pot fi abordate într-un mod similar cu cele legate de transferurile de date.

Rata de transfer este cel mai important parametru de performanţă pentru un canal de comunicaţii, dar ea este limitată drastic de tehnologia utilizată. Rezultă că, pentru a depăşi aceste limite, singurele soluţii sunt cele structurale, care presupun utilizarea în paralel a mai multor seturi de canale, inclusiv emiţătoarele şi receptoarele aferente. Creşterea numărului de componente va conduce, însă, la o scădere atât a fiabilităţii cât şi a disponibilităţii. Pentru a depăşi aceste probleme suplimentare propun în continuare utilizarea unor soluţii bazate pe redundanţă şi codificare.

Pentru transferuri în care datele care pot fi divizate în pachete sau blocuri protejate prin coduri detectoare de erori voi prezenta în continuare o soluţie originală de creştere a fiabilităţii şi performanţelor transferurilor de date. Această soluţie am denumit-o ***RAIC (Redundant Array of Independent Channels).***

Pentru a creşte rata de transfer, datele pot fi distribuite simultan prin mai multe canale independente. Pentru un număr *n* rezonabil de astfel de canale se poate obţine o rată globală de transfer de *n* ori mai mare decât în cazul utilizării unui singur canal. Desigur un număr de canale pot fi rezervate pentru redundanţa în scopul acoperirii unor eventuale erori.

În Figura 5-3 prezint o sugestie de utilizare a unor coduri pentru câteva tipuri particulare de canale:

TIP DE CANAL	COD RECOMANDAT
MEMORII RAPIDE (la nivel de bit)	HSIO (SEC-DED) HAMMING MODIFICAT (SEC-DED) IMAI- KAMIYANAGI HORIGUCHI- MORITA PARITATE INCRUCISATA
MEMORII RAPIDE (la nivel de cuvânt)	HAMMING BURTON FUJIWARA HONG-PATEL REED-SOLOMON KANEDA-FUJIWARA CHEN
DISCURI	FIRE REED-SOLOMON INTRETESUT (SI INCRUCISAT)
MEMORII CU BULE	HAMMING FIRE
PROCESOR	PARITATE BERGER

Figura 5-3

Din păcate, RAIC nu reduce costul transferurilor de date, dar asigură o creştere spectaculoasă a performanţei şi a fiabilităţii acestora.

În continuare, voi analiza principalele probleme legate de **RAIC,** cum ar fi: organizarea şi distribuţia datelor, adresarea independentă, divizarea fină, divizarea grosieră, alegerea mărimii unităţii de divizare, mecanisme de redundanţă, replicarea datelor, protecţia bazată pe paritate.

5.2 Construirea sistemului RAIC

În cele ce urmează, vom explica cum putem genera octeţii de control, precum şi cum îi vom folosi pentru a implementa operaţia de siguranţă la apariţia unui defect în sistemul de intrare/ieşire. Să presupunem că avem trei canale şi pe fiecare dintre ele transferăm un octet (adică opt biţi) de date. Două vor transfera informaţie utilă, în timp ce al treilea va transfera un octet de control, generat cu ajutorul funcţiei SAU exclusiv (XOR).

În cazul în care canalul numărul 1 are transferat octetul 00110000, iar canalul numărul 2 are transferat octetul 00000011, atunci octetul de control transferat în canalul numărul 3 este generat realizând operaţia

XOR asupra celor doi octeţi transferaţi în canalele 1 şi 2. Operaţia executându-se la nivel de bit, rezultă octetul 00110011. Transferând acest octet de control prin cel de-al treilea canal, putem să reconstituim data transferată prin oricare dintre cele trei canale, folosindu-ne de informaţia transferată în celelalte două. Astfel, presupunând că eşuează canalul numărul 1, data de pe acesta se reconstruieşte realizând un SAU exclusiv între octeţii dispozitivelor 2 şi 3:

```
  00000011   CANAL 2 (CANAL PENTRU DATE)
XOR 00110011 CANAL 3 (CANAL PENTRU PARITATE)
  00110000   CANAL 1 (DATĂ RECONSTRUITĂ)
```

Această tehnică se aplică pentru orice dimensiune a blocurilor, în mod identic. De asemenea, numărul de canale care intră în componenţa sistemului nu este limitat din punct de vedere teoretic.

Această caracteristică de siguranţă la defect se pretează sistemelor cu misiuni critice. Chiar dacă un canal din set se defectează, sistemul continuă să fie operaţional. Oricum, canalul defect va putea fi înlocuit imediat.

În cele ce urmează voi folosi următoarea convenţie:

- scriere – operaţie de transfer la nivelul emiţătorului;
- citire – operaţie de transfer la nivelul receptorului.

5.3 Aranjamentul octeţilor de control

Până acum am arătat cum oferă un sistem RAIC o performanţă ridicată împrăştiind datele, modul în care disponibilitatea datelor este îmbunătăţită utilizând funcţia XOR pentru generarea octeţilor de control, precum şi modul în care se reconstruieşte informaţia unui canal care a eşuat. Pentru a completa această analiză, iată cum se poate aranja informaţia utilă pe canale, pentru obţinerea unei funcţionări mai eficiente.

Pentru a avea tot timpul actualizată informaţia de control, octeţii de control trebuie să fie modificaţi de fiecare dată când se transferă ceva pe oricare dintre canale. Dacă proiectăm sistemul în aşa fel încât toţi octeţii de control să fie transmişi pe acelaşi canal, se poate crea o gâtuire a canalului, rezultând o scădere a performanţei sistemului RAIC.

Pentru a evita această problemă este mult mai eficient să transmitem pe fiecare canal atât date utile, cât şi octeţi de control. Împrăştiind octeţii de control peste toate canalele din sistem, se permit transferuri multiple simultane. Într-un sistem RAIC cu patru canale, acesta oferă o triplare a performanţelor, în ceea ce priveşte operaţiile de transfer de blocuri mici de date, în comparaţie cu un sistem în care toată informaţia de control este transmisă pe un singur canal. Cu cât numărul de canale creşte, performanţa sistemului creşte proporţional.

În continuare, prezint un exemplu în care am un sistem RAIC cu patru canale, pe fiecare canal transmiţându-se patru blocuri (nu interesează mărimea blocului, dar ea trebuie să fie aceeaşi pentru toate cele patru canale). În total avem 16 blocuri, dintre care în 12 blocuri se află informaţie utilă, iar în 4 blocuri avem informaţie de control (notată Cb(a-b-c), unde *a*, *b* şi *c* sunt numerele blocurilor de date asociate acestui bloc de control):

Canal 1	Canal 2	Canal 3	Canal 4
Bloc 1	Bloc 2	Bloc 3	Cb(1-2-3)
Bloc 5	Bloc 6	Cb(4-5-6)	Bloc 4
Bloc 9	Cb(7-8-9)	Bloc 7	Bloc 8
Cb(10-11-12)	Bloc 10	Bloc 11	Bloc 12

Figura 5-4

5.4 Organizarea datelor

Organizarea datelor pentru matricele de canale poate fi partiţionată în două componente ortogonale: schemele de distribuţie a datelor şi mecanismele de redundanţă. Distribuţia de date defineşte translaţia adreselor logice, vizibile extern, în locaţii de transfer din cadrul subsistemului. Sunt două căi pentru a face acest lucru: prin adresarea independentă a fiecărui canal sau prin divizarea canalului. Cea din urmă oferă, adesea, grade îmbunătăţite de încărcare echilibrată, transfer paralel de date sau acces concurent. Mecanismele de redundanţă specifică tipul, scopul şi locaţia oricărei informaţii redundante din matrice (de exemplu, replicarea datelor şi paritatea divizată). Matricele redundante de canale continuă să ofere acces deplin la date şi după

defectarea unui canal şi în timp ce datele pierdute sunt reconstruite pe un canal de înlocuire sau din informaţiile recepţionate corect. Matricele de canale pot combina divizarea canalului cu redundanţa pentru a oferi atât performanţă cât şi fiabilitate.

5.4.1 Distribuţia datelor

Distribuţia de date constă în maparea adreselor logice folosite de sistemul gazdă pe canalele din subsistem. Există, în principiu, trei metode principale folosite în realizarea acestei translaţii. Metoda convenţională este de a adresa, independent, fiecare canal şi de a mapa numerele de bloc logic în numere de bloc de canal, direct. Distribuţia este făcută manual de către administratorii de sistem, programe de aplicaţii sau sisteme *software*. Divizarea de canal, numită şi *channel striping* sau *channel interleaving*, reuneşte mai multe spaţii de adrese de canal într-un singur spaţiu, unificat, văzut de sistemul gazdă. Aceasta se face prin distribuirea consecutivă de unităţi logice de date (unităţi de divizare) din cadrul canalului într-o manieră *round-robin*. Se disting între două tipuri diferite de divizări de canale: divizare fină, în care canalele cooperează toate pentru satisfacerea unei cereri şi divizare grosieră, în care canalele cooperează la cereri mari de date şi lucrează independent la cereri mici de date.

5.4.1.1 Adresarea independentă

Dacă fiecare canal este adresat independent, utilizatorii sau sistemele gazdă trebuie să distribuie explicit datele. În general, administratorii de sistem sunt responsabili pentru răspândirea datelor pe fiecare canal. Încărcarea echilibrată sau distribuirea datelor astfel încât să se încarce echilibrat sistemul, este mai mult o artă decât o ştiinţă şi necesită specialişti. Adrese de date adiacente creează *hot spots* făcând *task*-ul complex.

Canal 0	Canal 0	Canal 3	Canal 4
A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
...	...	...	...

Figura 5-5

5.4.1.1.1 Divizarea fină

Aici, toate cele *n* canale conţin o fracţiune din fiecare bloc accesibil (Figura 5-6). Numărul canalelor şi mărimea unităţii de divizare sunt, în general, alese astfel încât produsul lor să dividă egal cea mai mică unitate de date accesibilă. Mărimi mai folosite sunt un bit, un octet, secţiune. Mărimea folosită nu este neapărat importantă deoarece fiecare unitate accesibilă este răspândită pe toate canalele. Încărcarea este echilibrată deoarece toate canalele primesc aceeaşi încărcare. Rata de transfer este de aproximativ *n* ori mai mare faţă de cea a unui canal individual deoarece fiecare canal transferă $1/n$ din datele cerute. De asemenea, doar o singură cerere poate fi făcută la un moment dat, deoarece toate cele *n* canale lucrează la aceeaşi cerere. Aceste limitări, de obicei, restrâng folosirea metodei la domenii în care timpii de transfer domină timpii în care nu se lucrează cu canalul.

Canal 0	Canal 0	Canal 3	Canal 4
A0	A0	A0	A0
A1	A1	A1	A1
...	...	...	...
A7	A7	A7	A7
...	...	...	...
B0	B0	B0	B0
B1	B1	B1	B1
...	...	...	...

Figura 5-6

5.4.1.1.2 Divizarea grosieră

Cu aceasta nu mai este necesar ca toate canalele să coopereze la fiecare cerere. Această tehnică exploatează paralelismul transferului de date pentru cereri mari, în timp ce permite canalelor separate să satisfacă cereri mici concurent. Câteva efecte se combină pentru a oferi o încărcarea echilibrată şi automată. Primul efect este acela că un fişier va tinde să aibă blocurile sale componente împrăştiate pe mai multe canale. Divizarea grosieră, ca la *hashing* efectiv, randomizează primul canal accesat pentru fiecare cerere, celelalte canale fiind identificate în maniera *round-robin* (Figura 5-7). Această încărcare, echilibrată statistic, rămâne intactă şi în cazul adreselor de date adiacente şi distribuţii rapid schimbătoare de accese. Din motivele de mai sus, metoda are posibilitatea să ofere performanţă bună cu un minim de întreţinere.

Canal 0	Canal 0	Canal 3	Canal 4
A0	A1	A2	A3
A4	A5	A6	A7
...	...	...	...
B0	B1	B2	B3
...	...	...	...

Figura 5-7

5.4.1.1.3 Alegerea mărimii unităţii de divizare

Realizarea performanţei de la metoda de mai sus necesită alegerea corectă a mărimii unităţii de divizare. Această alegere determină, pe de o parte, numărul de canale peste care sunt răspândite datele fiecărei cereri. Mărimi mari ale unităţii permit canalelor separate să satisfacă cereri multiple (concurent), rezultând adesea reducerea cozilor de aşteptare şi o rată de transfer mai mare. Pe de altă parte, unităţile mici determină mai multe canale să acceseze datele în paralel, reducând timpii de transfer pentru cereri individuale. Acest compromis între paralelismul transferului şi concurenţa accesului este guvernată de mărimea unităţii de divizare.

Când avem cereri de lungime variabilă, fără o secvenţializare sau localizare a adreselor, parametrul cheie este cantitatea de concurenţă din încărcare. Mărimea unităţii de divizare ar trebui să crească o dată cu rata de sosire a cererilor, deci descrescând numărul mediu de canale accesate pe cerere.

Pentru cereri constând în primul rând din cereri aliniate, de mărime fixă, mărimea unităţii de divizare poate fi aleasă astfel încât numărul de accese la canal, pe cerere, să fie constant. În particular, folosind orice multiplu de mărimea fixă a cererii, se permite fiecărei cereri să fie servită de către un singur canal.

Pentru încărcări cu un oarecare grad de aliniament sau localizare a adresei, echilibrarea trebuie, de asemenea, considerată. Când cererile sunt mari, echilibrarea nu afectează semnificativ alegerea mărimii unităţii de divizare. Pentru încărcări mici, unităţile mari exploatează

beneficiile timpului de acces datorate localizării şi secvenţialităţii adreselor.

Un compromis mai complex este cel pentru încărcări generale, adesea caracterizate prin cereri mici, cereri mari scurte, şi adrese puternic adiacente. În aceste cazuri, abilitatea metodei divizării grosiere de a echilibra încărcarea se îmbunătăţeşte pe măsură ce mărimea unităţii de divizare scade. Mărimi mai mari ale unităţii tind să sufere din cauza formării unor *hot spots* de scurtă durată. Sugerez ca unitatea de divizare trebuie să fie de 10 ori mai mare decât mărimea medie a cererii, rotunjită la un multiplu al mărimii blocului. Alegerea mărimii unităţii de divizare poate fi rafinată considerând informaţie adiţională, cum ar fi distribuţiile de mărime a cererii şi caracteristicile localizării.

O întrebare interesantă este dacă mărimea unităţii de divizare trebuie să fie multiplu de mărime a blocului. Pe de o parte, o cerere pentru unitatea de divizare completă are avantajul unui canal suportând acces cu latenţă zero. Pe de altă parte, mărimea blocului poate corespunde, în mod dificil, mărimii şi aliniamentului cererii, rezultând cereri multicanal, excesive. Multe încărcări constau din cereri de blocuri de mărime specifică, să zicem 4k sau 8k. Cele mai multe unităţi de canal *state-of-the-art* folosesc transferuri pe zone multiple pentru a creşte densitatea de transfer. Folosind aceste canale, potrivirea mărimii unităţii de divizare cu cea a blocului cere o mărime diferită pentru fiecare zonă de canal.

5.4.2 Mecanismele de redundanţă

Adăugând mecanisme de redundanţă la un sistem de transfer, invariabil îi creşte costul, i se reduce capacitatea şi/sau i se reduc performanţele. Din fericire, căderile individuale de canale sunt foarte rare, iar căderile de sistem sunt şi mai rare. Timpul mediu până la defectare (MTBF) la un canal modern este de sute de mii de ore, altfel spus, zeci de ani.

Totuşi, pe măsură ce numărul de canale creşte, MTBF-ul unui sistem tipic neredundant se micşorează până la perioade de luni sau chiar săptămâni. Presupunând că MTBF-ul fiecărui canal este independent şi exponenţial distribuit, MTBF-ul unui grup este invers proporţional cu numărul canalelor. Aceasta ameninţă fezabilitatea sistemelor nonredundante de canale.

În plus, numărul de canale afectate de cădere creşte dramatic o dată cu folosirea divizării canalelor. Un subsistem divizat poate pierde $1/n$ din fiecare fişier, faţă de $1/n$ fişiere din matricea de canale în cazul adresării independente. Un subsistem cu redundanţă în organizarea datelor poate

supravieţui căderii uneia sau mai multor componente. Totuşi, la apariţia unor noi căderi va rezulta inaccesibilitatea sau pierderea datelor. O dată cu creşterea subsistemelor de transfer, redundanţa *on-line* va fi necesară pentru datele cruciale, devenind, probabil, un standard pentru transfer, în general.

5.4.2.1 Replicarea datelor

Cel mai simplu mod de a proteja datele este replicarea lor. Copii separate ale fiecărui bloc sunt transferate pe canale separate. Datele devin indisponibile doar dacă toate canalele, conţinând o copie, cad. Această formă de redundanţă, folosită de mai mulţi ani, se numeşte *channel mirorring*, *channel duplexing*, *channel shadowing* sau, mai nou, RAIC 1. Costul replicării datelor este proporţional cu *c-1* (*c* fiind numărul de canale din matrice), deoarece costul canalului domină preţul subsistemelor de transfer. Deoarece două copii oferă înaltă fiabilitate, alte copii, în plus, nu se folosesc. De fapt, costul mare al replicării obligă, adesea, la folosirea altor mecanisme de redundanţă care nu dau acelaşi nivel de performanţă şi fiabilitate.

Utilizarea unor copii multiple ale datelor afectează performanţa în diferite moduri. Primul este acela că scrierile trebuie făcute pe toate copiile. Depinzând de implementare, aceasta poate crea oportunităţi pentru pierderea datelor sau timpi crescuţi de răspuns. Performanţa la citire este, pe de altă parte, mai bună în acest caz. În primul rând, cererile de citire se satisfac simultan, crescând rata de transfer şi scăzând timpii de aşteptare. În plus, o cerere de citire poate fi programată pe canalul cu cel mai mic timp de acces. Această performanţă bună la citire poate face raportul cost/performanţă chiar mai mic decât în cazul sistemelor cu canale nonredundante, deşi costul absolut este de *c* ori mai mare.

Metoda convenţională de implementare a replicării datelor dublează fiecare canal (Figura 5-8). Fiecare set de *c* canale identice pot supravieţui la *c*-1 căderi fără pierdere de date. Într-o matrice cu seturi multiple replicate, pot apărea multe căderi, fără să cauzeze vreo pierdere de informaţii, atâta timp cât *c* din ele nu sunt din acelaşi set unic, replicat. Totuşi, fiecare cădere de canal reduce performanţele unui set relativ la alte seturi. Acest dezechilibru poate duce la scăderea performanţelor generale ale sistemului, dacă un set căzut devine o gâtuire.

Alternativ, seturile se pot suprapune astfel încât această degradare de performanţă este împărţită în mod egal de toate canalele rămase. Termenul *declustering* se referă la o metodă pentru combinarea seturilor multiple replicate. Metoda *chained declustering* se referă la divizarea

fiecărui canal în *c* secţiuni. Prima secţiune conţine copia primară a unor informaţii, cu fiecare canal având date primare diferite (a se vedea Figura 5-9). Cu o aranjare inteligentă, această metodă poate menţine o încărcare echilibrată după o cădere a unui canal, deşi probabilitatea supravieţuirii la căderi multiple este redusă.

Al doilea tip de *declustering*, cunoscut şi sub numele de *double declustering* (vezi Figura 5-10) partiţionează, de asemenea, fiecare canal, dar divizează copiile nonprimare din toate canalele în loc să le păstreze în spaţii contigue. Aceasta echilibrează încărcarea suplimentară cauzată de o cădere de canal, dar reduce, în continuare, fiabilitatea (pierderea oricăror *c* canale rezultă într-o cădere critică).

Canal 0	Canal 1	Canal 2	Canal 3
0	0	2	2
1	1	3	3

Figura 5-8

Canal 0	Canal 1	Canal 2	Canal 3
0	1	2	3
3	0	1	2

Figura 5-9

Canal 0	Canal 1	Canal 2	Canal 3
0	1	2	3
1c	0a	0b	0c
2b	2c	1a	1b
3a	3b	3c	2a

Figura 5-10

5.4.2.2 Protecţia bazată pe paritate

Proiectanţii de subsisteme de canale se pot folosi de rezultatele studiilor extinse ale codurilor detectoare şi corectoare de erori. Din teoria codurilor se ştie că pierderea unui singur bit poate fi recuperată ştiind bitul de paritate, ceilalţi biţi de protejat şi faptul că acel bit este eronat. Deoarece sistemele actuale de canale au o logică de control care, în general, detectează un procent semnificativ din căderi, aceasta poate fi folosită pentru protejarea unui set de canale faţă de erorile singulare. Fiecare bit de paritate este calculat din bitul de date asociat de pe fiecare canal protejat şi transferat pe un canal separat. Distribuirea unor biţi particulari de date daţi de un bit de paritate pot varia de la schemă la schemă, aşa cum poate varia şi aranjarea parităţii. Capacitatea adiţională cerută (un canal complet) pentru a menţine redundanţa prin intermediul parităţii este considerabil mai mică decât atunci când se foloseşte replicarea datelor. Deşi costul este rezonabil, menţinerea parităţii poate reduce în mod semnificativ performanţa. Cererile de citire sunt efectuate ca în matrice nonredundante. Dacă se adaugă un canal pentru a oferi capacitate de transfer pentru paritate, el va genera încărcare în plus pentru cereri de scriere (în cele mai multe scheme de aşezare a parităţii). Cererile de scriere necesită lucru suplimentar cu canalul pentru actualizarea parităţii, adesea ducând la o performanţă scăzută. Dacă toţi biţii de date asociaţi cu un bit de paritate sunt actualizaţi de o cerere de scriere, noul bit de paritate este calculat şi inclus ca o parte a cererii totale. Dacă doar un subset al biţilor de date este scris, actualizarea bitului de paritate necesită lucru suplimentar. Aceasta constă în citirea biţilor de pe canale pentru construirea noului bit de paritate. Avem două metode:

- **Read-modify-write,** unde noul bit de paritate este construit din valorile vechi şi cele noi ale biţilor care se scriu;
- **Regenerate write,** în care se generează bitul nou de paritate din valorile tuturor biţilor (adică noile valori ale biţilor care se scriu şi valorile curente ale biţilor neschimbaţi).

RMW necesită, mai întâi, accese de citire la toate canalele care se actualizează (incluzând canalul care conţine paritatea), pe când R-W cere accese de citire la toate canalele care nu se actualizează. Pierderea performanţei poate fi redusă prin combinarea dinamică a acestor două opţiuni. Datorită nevoii de a face citiri de pe canal înainte de actualizarea parităţii, performanţa poate fi semnificativ mai mică faţă de un sistem neprotejat sau faţă de un sistem protejat prin replicarea datelor.

Reducerea cât mai mult a acestei degradări de performanţă este, încă, un subiect aflat în cercetare.

Canal 0	Canal 1	Canal 2	Canal 3
A	A	A	a
B	B	B	b
C	C	C	c
...	...	...	...

Figura 5-11

Canal 0	Canal 1	Canal 2	Canal 3
A	A	A	a
B	B	b	B
C	c	C	C
...	...	...	...

Figura 5-12

Canal 0	Canal 1	Canal 2	Canal 3
A	A	A	a
B	b	C	B
B	C	c	C
...	...	...	...

Figura 5-13

Există trei scheme majore pentru împrăştierea informaţiei de paritate în cadrul canalelor din subsistem. Metoda directă este de a plasa toţi biţii de paritate pe un canal dedicat de paritate (Figura 5-11). Deşi simplifică

maparea adreselor logice în adrese de canal, fiecare scriere trebuie să actualizeze biţii asociaţi de pe unicul canal de paritate. Matricele care folosesc divizarea fină sunt potrivite pentru această metodă, deoarece ele satisfac doar o cerere la un moment dat. Cu alte scheme de distribuţie, totuşi, canalele pot limita performanţa, mai ales când vechea paritate este cerută să participe în operaţiile de la RMW. Pentru evitarea acestei potenţiale gâtuiri, informaţia de paritate poate fi divizată printre canale (vezi Figura 5-12). Folosind divizarea parităţii, matricea poate efectua mai multe actualizări în paralel.

O a treia opţiune propusă: *declustered parity*. Pentru înţelegerea acestei scheme ne putem imagina combinarea câtorva matrice mai mici, protejate de paritatea divizată. Datele şi paritatea fiecărei matrice sunt distribuite în întregul set de canale disponibile (a se vedea Figura 5-13). Cu un algoritm inteligent de distribuţie, impactul performanţei datorat pierderii unui canal poate fi împărţit, în mod egal, de toate canalele rămase. Deşi matricea poate supravieţui doar unei singure căderi, ar trebui să se refacă mai repede. Ca o alternativă, aceasta metodă se poate imagina ca o cale de a realiza fiabilitate mai bună şi performanţă în funcţionarea la defecte prin creşterea cantităţii de date de paritate menţinută.

5.4.2.3 Alte scheme

Menţinerea parităţii protejează o matrice împotriva căderii unui singur canal. O matrice cu două copii ale tuturor datelor poate supravieţui până la o cădere pe perechea de copii. În ambele cazuri, datele pot fi pierdute dacă apare o a doua cădere, înainte ca prima să fie reparată. Oferirea protecţiei necondiţionate împotriva căderilor multiple necesită redundanţă suplimentară. În cele mai promiţătoare metode se folosesc două coduri *Reed-Solomon*, corectoare de erori. Cu asemenea redundanţă, o matrice poate reface toate datele după pierderea accesului la oricare două canale. Compromisurile referitoare la performanţa şi opţiunile de amplasare a redundanţei se potrivesc pentru acele matrice cu protecţie bazată pe paritate, în afară de faptul că doi biţi sunt mai bine menţinuţi decât unul.

Alte scheme, cum ar fi paritatea multidimensională şi codurile mai agresive, corectoare de erori, pot proteja împotriva unui număr mai mare de căderi. Totuşi, costul şi implicaţiile de performanţă, combinate cu faptul că fiabilitatea sistemelor de transfer creşte tot mai mult, pot obliga folosirea lor.

5.5 Performanţa versus fiabilitatea

Cererile de scriere pot duce la întârzieri mari între momentul iniţierii lor şi momentul scrierii efective a noii informaţii pe canal. Până când sistemul gazdă primeşte mesajul de satisfacere a cererii orice resurse de sistem nu pot fi reutilizate. De asemenea, procesele de aplicaţie pot aştepta pentru astfel de confirmări, putând rezulta cicluri inutile de procesor. Acceptând confirmarea cât mai repede posibil, gazda poate utiliza mai bine resursele sale şi are, astfel, mai puţine cicluri inutile de procesor, rezultând o performanţă mai bună. Totuşi, în cazul în care confirmarea cererii este dată înainte ca datele să fie scrise efectiv pe canal, sistemul este vulnerabil la coruperea sau pierderea informaţiilor, datorită căderii tensiunii sau defectării componentelor. Când sistemul renunţă la constrângerile de fiabilitate în scopul câştigării performanţei, trebuie luate măsuri de precauţie sau se acceptă consecinţele.

5.5.1 Negocierea fiabilităţii în favoarea performanţei

Un controler de matrice de canale cu un *cache* (sau *buffer*) poate semnaliza confirmarea gazdei de îndată ce ultimul octet a fost copiat din memoria principală. Astfel de scheme îmbunătăţesc mult performanţa văzută de gazdă la scriere, deşi datele sunt vulnerabile până la scrierea pe canal. Managerul de canal, care este responsabil de determinarea ordinii de servire pentru accesele la canal aflate în aşteptare, are, acum, o problemă suplimentara. Fiabilitatea cere managerului să expedieze accesele la canal care transferă datele din *cache* pe suportul fizic, în timp ce performanţa obligă managerul să amâne această activitate de fundal.

În matricele care folosesc replicarea informaţiilor, raportarea confirmării după ce doar o fracţiune din copii a fost actualizată poate şi ea să îmbunătăţească performanţa. Datele sunt pierdute doar dacă apar căderi multiple înainte să se efectueze scrierile rămase. Pentru matricele protejate prin paritate, punând paritate în *cache* poate să se îmbunătăţească performanţa prin acumularea noii parităţi pentru scrieri multiple şi efectuarea actualizări de paritate într-o singură scriere.

În schemele de paritate ce cer cicluri de actualizare, sistemul fiind vulnerabil la coruperea datelor dacă setul de scrieri pe canal nu se face simultan. În cazul în care cade tensiunea, la un moment nepotrivit, doar câteva cereri vor fi satisfăcute. Aceasta face ca paritatea să fie incorectă, ducând la coruperi ulterioare dacă sunt lăsate necorectate şi apoi folosită la refacerea datelor. Pe de altă parte, impunerea scrierilor simultane necesită comportare sincronă din partea mai multor canale. Aceasta

poate produce programări neoptime ale cererilor şi un timp de canal inutil crescut. Când controlerul îşi programează, independent, cererile individuale de scriere utilizarea canalelor poate fi maximizată.

5.5.2 Îmbunătăţirea performanţei şi a fiabilităţii

Cei mai mulţi utilizatori nu sunt deranjaţi de coruperea datelor, exceptând cazurile inacceptabile. Există câteva tehnici pentru îmbunătăţirea performanţei şi a fiabilităţii. Ele se realizează prin reducerea latentelor la scriere în timp ce este garantată siguranţa datelor vechi şi se asigură securitatea datelor noi înainte de semnalizarea confirmării. Aceste scheme pot fi implementate independent de sistemul gazdă sau în cadrul matricei de canale. O soluţie simplă este folosirea memoriei RAM *cache*, non-volatilă, pentru a scrie datele sau paritatea. Aceste memorii menţin datele chiar şi după întreruperea tensiunii de alimentare. Această organizare ar trebui să ofere o fiabilitate echivalentă cu cea a unei scheme cu redundanţă. Remaparea scrierii, o formă de paginare dublă, poate fi utilizata pentru multe organizări de date diferite. Această remapare constă în alterarea dinamică a mapării adreselor logice în cele fizice, astfel încât cererea de scriere de date este totdeauna plasată la locaţii noi. Prin menţinerea copiei anterioare a blocului logic până la satisfacerea cererii, metoda îmbunătăţeşte fiabilitatea. Desigur, mapările trebuie menţinute fiabil pentru a preveni pierderea informaţiilor. O metodă este transferarea informaţiei în NVRAM. Altă metodă, mai complexă, dar mai puţin scumpă este să mărim fiecare bloc fizic de date cu adresa logică corespunzătoare şi cu un indicator de timp, care pot fi folosite să reconstruiască mapările.

Algoritmul pentru alegerea locaţiei disponibile trebuie să prevină consumarea tuturor locaţiilor libere dintr-o anumită regiune şi lăsarea altor regiuni nefolosite. De asemenea, în timp, datele secvenţial logice vor fi disparate pe spaţiul fizic.

A treia opţiune - *logging* - scrie informaţia despre actualizări într-o zonă de log înainte de actualizările propriu-zise. Scrierile log sunt secvenţiale şi au, de obicei, un timp de răspuns mai mic, mai ales când se foloseşte un canal log dedicat. După scrierea intrării de log, controlerul de matrice poate, în siguranţă, să transmită un mesaj de terminare către gazdă şi gazda accesează canalul în fundal. În caz de cădere, conţinutul log-ului permite repetarea oricărei actualizări incomplete. Informaţia scrisă în log poate să conţină o descriere scurtă a actualizării, o copie a noii informaţii sau copii ale amândurora. O formă de verificare este folosită pentru a reduce timpul de refacere şi permite refolosirea spaţiului de log.

5.5.3 Tratarea căderilor

Prin menţinerea redundanţei, o matrice de canale poate supravieţui căderii unuia sau mai multor canale. În timp ce matricea continuă să ofere acces la toate datele, performanţa ei este aproape întotdeauna afectată de căderi. În plus, fiabilitatea este redusă, deoarece căderile ulterioare pot fi critice. După cădere, datele pierdute trebuie să fie reconstruite de pe canalele rămase şi eventual scrise pe un canal de înlocuire, proces numit refacere. Până când acest proces se termină, matricea este vulnerabilă. Canalele suplimentare, numite rezerve rapide, sunt adesea incluse în matrice ca să permită refacerii să înceapă imediat.

În cele mai multe scheme, pierderea accesului la canal reduce numărul de cereri concurente de citire care pot fi deservite. Matricele care folosesc replicarea de date primesc cereri ca de obicei, dar mai puţine canale efectuează citiri. De asemenea, scrierea necesită actualizarea unei copii mai puţin. La matricele protejate prin paritate, o cerere de citire spre un canal defect cere o regenerare a datelor lipsă. Fiecare bit de date pierdut poate fi refăcut din bitul de paritate asociat şi din ceilalţi biţi de date. Deci, lucrul cu cereri de citire necesită acces la toate canalele care conţin aceşti biţi. Performanţa la scriere este şi ea afectată în matricele protejate de paritate. Datorită faptului că este afectată şi vechea informaţie, *Regenerate-Write* trebuie folosit pentru toate actualizările de date pe canalul căzut şi RMW pentru toate actualizările pe canalele rămase.

Cea mai simplă formă de refacere reconstruieşte secvenţial informaţiile indisponibile şi le scrie pe un canal de înlocuire. Într-o matrice replicată, datele pierdute sunt copiate de pe un alt canal conţinând aceleaşi date. Într-una cu protejare bazată pe paritate datele trebuie refăcute din biţii de paritate şi cei de date. Dacă cerinţele de fiabilitate prevăd ca timpul de reparare să fie minim, cererile gazdei pot fi anulate în timp ce refacerea are loc la viteză maximă. Altfel, refacerea şi sistemul gazdă concurează pentru deservire, rezultând un alt compromis între fiabilitate şi performanţă. Acest compromis este, în particular, important în refacerea matricelor cu protecţie bazată pe paritate, unde regenerarea datelor cere accese la mai multe canale. Alegerea unui algoritm de refacere, mărimea unei cereri individuale de refacere, şi complexitatea circuisticii de coordonare multicanal trebuie luate în consideraţie când se ajustează procesul de refacere.

Dacă subsistemele de transfer continuă să se dezvolte ca şi până acum, nivelul inteligenţei încorporate în diferitele lor componente va continua să crească. Comunicaţia între aceste componente va creşte mult,

rezultând optimizări globale, un control mai distribuit, şi noi algoritmi de distribuţie care răspund dinamic la mereu schimbătorul flux de cereri.

5.6 Propunere de clasificarea pentru RAIC

În continuare, voi propune o clasificare a soluţiilor RAIC, pornind de la criterii funcţionale şi de la ideea considerării RAID ca o subclasă RAIC.

Diferitele niveluri de RAIC vor oferi diferenţe în performanţă, cost, asigurarea integrităţii şi a disponibilităţii datelor.

RAIC de nivel 0 - date distribuite fără utilizarea parităţii

RAID 0 presupune distribuirea datelor pe mai multe canale fără a utiliza nici un fel de redundanţă.

RAIC de nivel 1 – date duplicate

În RAIC 1 se utilizează mai multe seturi de date duplicate transferate pe canale separate. Dintre nivelurile de RAIC, acesta asigură cea mai mare disponibilitate a datelor, precum şi cea mai mare performanţă în transferul de date.

RAIC de nivel 0+1 – date distribuite şi replicate

Acest nivel intermediar combină avantajele RAIC 0 şi RAIC 1

RAIC de nivel 2 – protecţie prin coduri Hamming

În RAIC 2 datele sunt distribuite pe un număr de canale separate, la care se adaugă un număr de alte canale care să asigure corecţia unei erori de bloc, precum şi detecţia unei erori duble.

RAIC de nivel 3 – distribuţia datelor şi utilizarea unui canal separat pentru paritate

Datele sunt distribuite peste un număr de canale separate, la care se adaugă un canal pe care se transferă informaţia de paritate calculată cu ajutorul funcţiei XOR.

RAIC de nivel 4 – canale independente pentru distribuţia datelor şi paritate

Idem RAIC 3 cu observaţia că se pot utiliza şi independent canalele.

RAIC de nivel 5 – date şi paritate distribuite

Idem RAIC 4 cu observaţia că şi paritatea este distribuită pe un set dintre canalele matricei.

RAIC de nivel 6 – date şi paritate dublă distribuite

RAIC de nivel 7– RAIC 5 cu canale de rezervă active

5.7 Aplicarea conceptului RAIC în transmisiile seriale de date

Pentru a testa parțial conceptul de RAIC, am folosit un stand de încercare format din:

- PC AT PENTIUM II 450 Mhz dotate cu cate 8 interfețe seriale RS232c – 2 bucăți.
- un cablu pentru transfer date – 8*UTP de lungime 15m

Informația transferată a fost divizată în blocuri de câte 512 cuvinte, blocurile fiind protejate prin paritate încrucişată. După fiecare bloc transferat emițătorul confirma, cu un cuvânt de control, recepționarea corectă, corectată sau eronată.

Am folosit sisteme mult mai puternice decât ar fi fost necesare pentru aplicația dată în scopul reducerii la minimum a influențelor timpilor de calcul pentru paritate, respectiv pentru regenerarea datelor pierdute. Tot în acelaşi scop am utilizat paritatea încrucişată la nivel de blocuri.

Au fost testate, cu acest stand, RAIC 0, RAIC 1 *(chained declustering)*, RAIC 4, RAIC 5, RAIC 6, RAIC 7 (MFT 3).

RAIC 2 şi RAIC 3 le-am considerat nepotrivite pentru aplicația luată în considerație.

Interfețele seriale au fost setate pentru o rată de transfer de 33.6 Kbd.

Rezultatele obținute, în ceea ce priveşte rata globală de transfer, în condiții optime de funcționare, le prezint în Figura 5-14

Nivel RAIC	Rata obținută [Kbd]
RAIC 0	268.8
RAIC 1	134.4
RAIC 4	235.2
RAIC 5	235.2
RAIC 6	201.6
RAIC 7	168

Figura 5-14

Rezultatele obţinute, în ceea ce priveşte rata globală de transfer, în condiţiile în care am provocat o cădere fizică (tăiere o legătură), le prezint în Figura 5-15

Nivel RAIC	Rata obţinută [Kbd]
RAIC 0	0
RAIC 1	134.4
RAIC 4	201.6
RAIC 5	201.6
RAIC 6	168
RAIC 7	134.4

Figura 5-15

Rezultatele obţinute, în ceea ce priveşte rata globală de transfer, în condiţiile în care am provocat o a doua cădere fizică (tăiere o legătură), le prezint în Figura 5-16

Nivel RAIC	Rata obţinută [Kbd]
RAIC 0	0
RAIC 1	0 – 134.4
RAIC 4	0
RAIC 5	0
RAIC 6	134.4
RAIC 7	100.8

Figura 5-16

Rezultatele obţinute, în ceea ce priveşte rata globală de transfer, în condiţiile în care am provocat o a treia cădere fizică (tăiere o legătură), le prezint în Figura 5-17

Nivel RAIC	Rata obţinută [Kbd]
RAIC 0	0
RAIC 1	0 – 134.4
RAIC 4	0
RAIC 5	0
RAIC 6	0
RAIC 7	67.2

Figura 5-17

Rezultatele obţinute, în ceea ce priveşte rata globală de transfer, în condiţiile în care am supus unul dintre canale la influenţele unui puternic câmp variabil electromagnetic, le prezint în Figura 5-18

Nivel RAIC	Rata obţinută [Kbd]
RAIC 0	0
RAIC 1	134.4
RAIC 4	201.6
RAIC 5	201.6
RAIC 6	168
RAIC 7	134.4

Figura 5-18

Concluzii:

- cea mai bună performanţă a oferit-o RAIC 0, în condiţii normale de exploatare, dar şi cea mai slabă fiabilitate în condiţii defavorabile (stres fizic şi electromagnetic);
- cea mai sigură soluţie s-a dovedit RAIC 7.

5.8 Concluzii

Pe baza conceptelor prezentate, pe larg, în capitolul anterior, în partea a cincea a lucrării se prezintă o noua soluţie structurală de creştere a fiabilităţii sistemelor de calcul, soluţie care cuprinde ca pe un caz particular structurile RAID. Principalele contribuţii cuprinse în capitolul 5 sunt:

1) introducerea conceptului original – RAIC, care cuprinde complet, ca un subset, conceptul RAID;
2) definirea şi sistematizarea metodelor fundamentale de distribuţie a datelor într-un RAIC;
3) definirea şi sistematizarea principalelor mecanisme de redundanţă într-un RAIC;
4) o clasificare a structurilor RAIC;
5) realizarea practică a unei structuri RAIC.

6 CONCLUZII

Urmărind implementarea eficientă a toleranței la defectare în subsistemele I/O, precum şi creşterea performanțelor acestora prin noile structuri RAID level 7 şi RAIC, prezenta teză de doctorat aparține, în principal, domeniului fiabilității sistemelor de calcul.

Contribuțiile din cadrul tezei de doctorat au fost expuse, în extenso, prin cate un paragraf distinct, la sfârşitul fiecărui capitol. Aceste contribuții nu vor mai fi înşiruite, preferându-se o sinteză a celor mai importante, în acest capitol, jalonată de potențiale direcții de cercetare deschise. Teza cuprinde:

1) Contribuții la sistematizarea metodelor generale de creştere a fiabilității sistemelor de calcul;

2) Contribuții la definirea unor noi parametri cantitativi de evaluare a fiabilității subsistemului I/O;

3) Contribuții la sistematizare diferitelor structuri de discuri tolerante la defectare;

4) Dezvoltare domeniului matricelor de discuri RAID prin dezvoltarea unor algoritmi originali, precum şi prin definirea unei noi clase: RAID level 7;

5) Utilizarea generalizării noțiunilor de canal, transfer de date, emițători şi receptori, pentru reducerea formală a numărul de componente tipice ale unui sistem de calcul;

6) Fundamentarea teoretică a unui nou concept – RAIC – utilizabil în proiectarea sistemelor tolerante la defectare;

7) Realizarea practică a unei structuri RAIC.

Viitoarele posibile cercetări cu problematica cuprinsă în această lucrare ar putea fi orientate spre dezvoltări ale RAID level 7 şi, mai ales, în noul domeniu al RAIC.

7 BIBLIOGRAFIE

1. **[Adap92]** Adaptec Corporation: Selecting An Array Interface, Computer Technology Review, March 24, 1992, p. 27.
2. **[BBBM95]** M.Blaum, J.Brady, J. Bruck, J. Menon: EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. IEEE Transaction on Computers 44(2): 192–202, 1995.
3. **[Beal91]** D.G. Beal: Data Storage System for Providing Redundant Copies of Data in Different Disk Drives, International Patent Application, Publication Number WO 91/20034, December 26, 1991.
4. [**Bell84**] C.G. Bell: The Mini and Micro Industries Computer, vol. 17, no. 10, pp. 14-30, Oct. 1984.
5. **[Bell93]** Bellcore Applied Research Area: Profile of RAID — An Emerging Robust On-Line Storage Technology , Bell Communications Research publication, copyright 1992.
6. **[Bels91]** S.J. Belsan, et al: Deleted Data File Release Mechanism for a Dynamically Mapped Virtual Data Storage Subsystem, International Patent Application, Publication Number WO 91/20025, December 26, 1991.
7. **[BhDi92]** A. Bhide and D. Dias: RAID Architectures for OLTP, IBM Research Report RC 17879, March 24, 1992.
8. **[BiGr88]** D. Bitton and J. Gray: Disk Shadowing, Proc. 14th VLDB Conf., pp. 331–338, Los Angeles, Aug. 1988.
9. **[Bitt89]** D. Bitton: Arm Scheduling in Shadowed Disks, Proc. IEEE Spring ComCon, pp. 132–136, Feb. 1989.
10. **[BoCM91]** M.F. Bond, B.E. Clark and R.S. McRoberts: Method and Apparatus for Recovering Parity Protected Data, European Patent Application, Publication Number 0 462 917 A2, May 22, 1991.
11. **[Brad90a]** Brady, et al: Method and Means for Accessing DASD Arrays with Tuned data Transfer Rate and Concurrency, European Patent Application EU: 91304503.5, May 24, 1990.

12. **[Brad90b]** J.Brady: Organisation of XOR Hardware in Disk Arrays, IBM Invention Disclosure SA90-91-004, January 1990.

13. **[BuCS91]** W.A. Burkhard, K.C. Claffy, and T.J.E. Schwarz: Performance of Balanced Disk Array Schemes, Digest of Papers, 11th IEEE Symposium on Mass Storage Systems, October 1991.

14. **[BuJh94]** S.W. Burns, N.K. Jha: A Totally Sel-Checking Checker for a Parallel Unordered Coding Scheme. IEEE Transaction on Computers, April 1994, vol. 43, pp. 490–495.

15. **[Bult88]** D.L. Bultman: High Performance SCSI Using Parallel Drive Technology, Proc. BUSCON Conf., pp. 40–44, Anaheim CA, Feb. 1988.

16. **[CaLo91]** L-F. Cabrera and D.D.E. Long: SWIFT: Using Distributed Disk Striping to Provide High I/O Data Rates, Computing Systems, Fall 1991.

17. **[CARK91]** Chervenak, Ann L., Randy H. Katz: Performance of a RAID Prototype, 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, San Diego CA, May 1991.

18. **[CGGH96]** W.V. Courtright II, G.A. Gibson, M. Holland, J. Zelenka: RAIDframe: Rapid Prototyping for Disk Arrays. SIGMETRICS 1996: pp. 268–269.

19. **[CGKP90]** P. Chen, G. Gibson, R.H. Katz, and D.A. Patterson: An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890, Performance Evaluation Rev., vol. 18, no. 1, pp. 74–85, May 1990.

20. **[Chan92]** Chantal, A Division of BusLogic: Why Disk Arrays? Chantal Systems Publication, copyright 1992.

21. **[Chen91a]** C.H.Chen: Performance Models for Disk Arrays, IBM Tehnical Report TR-L41-C-3, 1991.

22. **[Chen91b]** C.H.Chen: Reliability of Some Reduntant Systems, IBM Technical Report TR-L41-C-1, 1991.

23. **[Chen91c]** C.H. Chen: M/G/1 Models of Disk Arrays with Small Data Transfer Size, IBM May 1991.

24. **[Chen92]** S.-Z. Chen: Design, Modeling, and Evaluation of High Performance I/O Subsystems, PhD thesis, Univ. of Massachusetts at Amherst, Sept. 1992.

25. **[Cher90]** A.L. Chervenak: Performance Measurement of the First RAID Protype, Technical Report UCB/CSD 90/674, UC Berkeley, May 1990.

26. **[Chie69]** R.T. Chien: Burst-Correcting Codes with High-Speed Decoding. IEEE Transaction on Information Theory IT-15 (January 1969): pp. 109–113.

27. **[ChKa91]** A.L. Chervenak and R.H. Katz: Performance of a Disk Array Prototype, CMG Transactions, Fall 1991.

28. **[ChKa91]** A.L. Chervanak, R.H. Kats: Performance of a RAID Prototyp. SIGMETRICS 1991: pp. 188–197.

29. **[ChLe95]** P.M. Chen, E.K. Lee: Striping in a RAID Level 5 Disk Array. SIGMETRICS 1995: pp. 136–145.

30. **[ChPa91]** P.M Chen and D.A. Patterson: Maximizing Performance in a Striped Array, CMG Transactions, Fall 1991.

31. **[ChRe92]** J.Chandy and A.L.N. Reddy: Failure Evaluation of Disk Array Organizations, IBM Research Report RC 8706, March 30, 1992.

32. **[ChTo92]** S.-Z. Chen and D. Towsley: Raid 5 vs. Parity Striping: Their Design and Evaluation, J. Parallel and Distributed Computing, vol. 17, pp. 58–74, 1992.

33. **[ChTo96]** S.Chen, D.F.Towsley: A Performance Evaluation of RAID Architectures. IEEE Transaction on Computers 45(10): pp. 1116-1130, 1996.

34. **[ChWH95]** A.J. Chaney, I.D. Wilson, A. Hopper: The Design and Implementation of a RAID 3 Multimedia File Server. NOSSDAV 1995: pp. 306–317.

35. **[Cipr90]** Ciprico, Inc.: Redundant Disk Arrays Enhance Data Safety To Support Network Servers, Computer Technology Review, Winter 1990, p. 20.

36. **[Clar88]** B.E. Clark, et al: Parity Spreading to Enhance Storage Acces, US Patent 4 761 785, August 1988.

37. **[CLDL94]** P.M. Chen, E.K. Lee, A.L. Drapeau, K. Lutz, E.L. Miller, S. Seshan, K. Shirriff, D.A. Patterson, R.H. Katz: Performance and Design Evaluation of the RAID 2 Storage Server. Distributed and Parallel Databases 2(3): pp. 243–260, 1994.

38. **[CLGK94]** P.M. Chen, E.L. Lee, G.A. Gibson, R.H. Katz, D.A. Patterson: RAID High-Performance, Reliable Secondary Storage. Computing Surveys 26(2): pp. 145–185, 1994.

39. **[CLVW93]** P. Cao, S.B. Lim, S. Venkataraman, J. Wilkes: The TickerTAIP Parallel RAID Architecture. ISCA 1993: pp. 52–63.

40. **[CoKe89]** G. Copeland and T. Keller: A Comparsion of High-Availability Median Recovery Techniques, ACM SIGMOD Conference, 1989.

41. **[CPDP90]** Chen, Peter M., David A. Patterson: Maximizing Performace in a Striped Disk Array, Proceedings of the 1990 ACM SIGARCH 17th Annual International Symposium of Computer Arhitecture, Seattle WA, May 1990.

42. **[CPGG90]** Chen, Peter M., Garth A. Gibson, Randy H.Katz, David A. Patterson: An Evalution of Redundant Arrays of Disks Using an Amdahl 5890, Proc. of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Boulder CO, May 1990.

43. **[Croc89]** T.W. Crockett: File Concepts for Parallel I/O, ACM 1989.

44. **[D&TR89]** D&T Roundtable: Built-in Self Tests: Are Expectations Too High? IEEE Design & Test of Computers, June 1989, pp. 66–74.

45. **[Data91]** DataQuest Disk Array Forecast, January, 1991.

46. **[Digi91]** DigiData: RAID Slow Getting Off The Ground, Computer Tehnology Review, Winter 1991, p. 44.

47. **[DoOu89]** F. Douglis and J. Ousterhout: Log-Structured File Systems, Proc. IEEE Spring ComCon, pp. 124–129, Feb. 1989.

48. **[DSHM94]** A.L. Drapeau, K. Shirriff, J.H. Hartman, E.L. Miller, S. Seshan, R.H. Katz, K. Lutz, D.A. Patterson, E.K. Lee, P.M. Chen, G.A. Gibson: RAID-II: A High-Bandwidth Network File Server. ISCA 1994: pp. 234–244.

49. **[Dunp90]** R.H. Dunphy Jr., et al: Disk Drive Memory, US Patent 4 914 656, April 1990.

50. [**Dyna92**] Dynatek Corporation: Implementing Raid in Cross-Platform Environments, Computer Technology Review, Winter 1992, p. 43.

51. **[Endl89]** ENDL Consulting: RAID Arrays Making a Slow Start, Computer Technology Review, September 1989, p. 22.

52. **[Fire59]** P.Fire: A Class of Multiple-Error-Correcting Binary Codes for Non-Independent Errors. Sylvania Report no. RSL-E-2, Mountain View CA: Sylvania Electronic Defense Laboratory, Reconnaissance System Division, 1959.

53. **[Form93]** Formation: Choosing Hardware/Software RAID, Computer Technology Review, July, 1993, p.39.

54. **[FRSH87]** A.W. Funkenbusch, T.A. Rinehart, D.W. Siitari, Y.S. Hwang, R.N. Gardner: Magneto-Optics Technology for Mass Storage Systems, May 1987: pp. 101–106.

55. **[Fuji90a]** E. Fujiwara: Logic Testing and Design for Testability. England: MIT Press, 1990.

56. **[Fuji90b]** E. Fujiwara, K. Pradhan: Error-Control Coding in Computers. IEEE Computer, July 1990, pp. 63–71.

57. **[Gait85]** N. Gaitanis: A Totally Self-Checking Error Indicator. IEEE Transaction on Computers, August 1985, vol. C-34, pp. 758–761.

58. **[Gait88]** N. Gaitanis: Totally Self-Checking Checkers with Separate Internal Fault Indication. IEEE Transaction on Computers, October 1988, vol. 37, pp. 1206–1213.

59. **[GaKo98]** E. Gabber, H.F. Korth: Data Logging: A Method for Efficient Data Updates in Constantly Active RAIDs. ICDE1998: pp. 144–153.

60. **[GaMS88]** H. Garcia-Molina and K. Salem: The Impact of Disk Striping on Realibility, IEEE Data Base Engineering Bulletin, March 1988.

61. **[GGHK89]** Gibson, Garth A., Lisa Hellerstein, Richard M. Karp, Randy H. Katz, David A. Patterson: Coding Techniques for Handling Failures in Large Disk Arrays Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III), Boston MA, April 1989, pp. 123–132.

62. **[GHKK89]** G.A. Gibson, L. Hellerstein, R.M. Karp, R.H. Katz and D.A. Patterson: Failure Correction Techniques for Large Disk Arrays, Proceedings the 3rd International Conference on Arhitectural Support for Programming Languages and Operating Systems, 1989.

63. **[Gibs89]** G.A. Gibson: Performance an Reliability in Redundant Arrays of Inexpensive Disks, CompCon, February 1989.

64. **[Gibs91]** G.A.Gibson: Redundant Disk Arrays: Reliable Parallel Secondary Storage, UC Berkeley Technical Report UCB/CSD 90/613, March 1991.

65. **[Gibs92]** Garth A. Gibson: Redundant Disk Arrays/Reliable, Parallel Secondary Storage, MIT Press, 1992.

66. **[Gibs95]** G.A. Gibson: Storage Technology: RAID and Beyond. SIGMOD Conference 1995: p. 471.

67. **[GiPa93]** G. Gibson and D. Patterson: Designing Disk Arrays for High Data Realibility, Journal of Parallel and Distributed Computing, January, 1993, pp. 4–27.

68. **[Gold91]** S. Goldstein: Array Storage Devices: Teaching Old DASD New Tricks, CMG Transaction, Fall 1991.

69. [**GrHW90**] J. Gray, B. Host, and M. Walker: Parity Striping of Disk Arrays: Low-Cost Reliable Storage with Acceptable Throughput, Proc. 16th VLDB Conf., pp. 148–161, Brisbane, Australia, 1990.

70. [**Hark81**] J.M. Harker et al.: A Quarter Century of Disk File Innovation, IBM J. Research and Development, vol. 25, pp. 677–689, Sept. 1981.

71. **[Hart88]** C.B. Hartness: Data Error Correction System, US Patent Number 4 775 978, October 4, 1988.

72. **[HCTS81]** M.Y. Hsiao, W.C. Carter, J.W. Thomas, W.R. Stringfellow: Reliability, Availability and Serviceability of IBM Computers Systems: A Quarter Century of Progress: IBM Journal of Research and Development 25 (September 1981): pp. 453–461.

73. **[HePa90]** J.L. Hennessy and D.L. Patterson: Computer Architecture — A Quantitative Approach. Morgan Kauffman, 1990.

74. **[Hida92]** HI-Data Corporation: One RAID Solution Does Not Fit All, Computer Technology Review, November 1992, p. 37.

75. [**HoGi92**] M. Holland and G. Gibson: Parity Declustering for Continuous Operation in Redundant Disk Arrays, Proc. ASPLOS-V, pp. 23–35, Sept. 1992.

76. **[HoPa93]** R.Y. Hou, Y.N. Patt: Comparing Rebuild Algorithms for Mirrored and RAID 5 Disk Arrays. SIGMOD Conference 1993: pp. 317–326.

77. **[Howe84]** T.D. Howell: Analysis of Correctable Errors in the IBM3380 Disk File: IBM Journal of Research and Development, March 28, 1984: pp. 206–211.

78. **[HsDe]** H. Hsiao and D. DeWitt: Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines, Proc. Sixth Int'l Conf. Data Eng., pp. 456–465, Los Angeles, Feb. 1990.

79. **[Ibis89]** IBIS Systems: Is a Disk Array a Cost-Effective Option? Computer Technology Review, May 1989, p. 22.

80. **[Inte92]** Interphase Corporation: Planing For Disk Array Integration, Computer Technology Review, November, 1992, p. 38.

81. **[IYHI87]** K. Itao, A. Yamaji, S. Hara, N. Izawa: Magneto-Optical Mass Storage System with 130 mm Write-Once Disk Compatibility: 8th IEEE Symposium on Mass Storage Syst. (May 1987): pp. 92–97.

82. **[John89]** B.W. Johnson. Design and Analysis of Fault Tolerant Digital System. Addison-Wesley Publishing Company, 1989.

83. **[John93]** B.W. Johnson. Boundary Scan Eases Test of New Technologies. Test&Measurement Europe, Autumn 1993.

84. **[KaGP89]** R.H. Katz, G. Gibson, and D.A. Patterson: Disk System Architectures for High Performance Computing, Proc. IEEE, vol. 77, No. 12, pp. 1842–1858, Dec. 1989.

85. **[KaVa89]** N. Kanopoulos, N. Vasanthavada: A Built-in Test Module for Fault Isolation. IEEE Design & Test of Computers, June 1989, pp. 58–65.

86. **[Kim85]** M.Y. Kim: Parallel Operation of Magnetic Disk Storage Devices: Synchronized Disk Interleaving ,Proceedings of the Fourth International Workshop on Data-Base Machines, 1985.

87. **[KiM86]** M.Y. Kim: Synchronized Disk Interleaving, IEEE Trans. on Computers, vol. 35, no. 11, pp. 978–988, Nov. 1986.

88. **[KiPa85]** M.Y. Kim and A. Patel: Error-Correcting Codes for Interveland Disks With Minimum Redundancy, IBM Research Report RC 11185, May 1985.

89. **[KiTa87**] M.Y. Kim and A. Tantawi: Asynchronous Disk Intervealing, IBM Reasearch Report RC 12497, January 1987.

90. **[KiTa91]** M.Y. Kim and A.N. Tantawi: Asynchronous Disk Interleaving: Approximating Acces Delays, IEEE Transaction on Computers, July 1991.

91. **[KiTa91]** M.Y. Kim and A.N. Tantawi: Asynchronized Disk Interleaving: Approximating Access Delays, IEEE Trans. on Computers, vol. 40, No. 7, pp. 801–810, July 1991.

92. **[KKIS87]** O. Kato, K. Kurosawa, K. Iwakuni, M. Shimbo: Development of Error Correction Method and LSI for Optical Disk Data Storage System: Paper of Technical Group on Computer Architecture, IPS Japan (September 1987): pp. 67–73.

93. **[KRGP89]** Katz, Randy H., G.A. Gibson, D.A. Patterson: Disk System Architectures for High Performance Computing, Proceedings of the IEEE, 77, December 12, 1989, pp. 1842–1858.

94. **[KRJO89]** Katz, Randy H., John K. Ousterhout, David A. Patterson, Peter M. Chen, Ann L. Chervenak, Rich Drewes, Garth A. Gibson, Ed K. Lee, Ken C. Lutz. Ethan L. Miller, Mendel Rosenblum: A Project on High Performance I/O Subsystems, ACM Computer Architecture News, 17, 5, Invited paper, September 1989, pp. 24–31.

95. **[Krol86]** T. Krol: (N,K) Concept Fault Tolerance. IEEE Transaction on Computers, April 1986, vol. C-35, No. 4, pp. 339–349.

96. **[Kurt87]** C. Kurtz : Development of a High-Capacity Performance Optical Storage System. Dig. 8th IEEE Symposium on Mass Storager System (May 1987): pp. 107–111.

97. **[Kurz88]** F. Kurzweil: Small Disk Arrays-The Emerging Approach to High Performance, CompCon, Spring 1988.

98. **[Lala85]** P.K. Lala: Fault Tolerant and Fault Testable Hardware Design. Prentice-Hall International, London, 1985.

99. **[Lawl81]** F.D. Lawlor: Efficent Mass Storage Parity Recovery Mechanism, IBM Technical Dislocure Bulletin 24, July 2,1978.

100. **[Lee90]** E.K.Lee: Software and Performance Issues in the Implementation of a RAID Prototype, Technical Report UCB/CSD 90/573, UC Berkeley, May 1990.

101. **[Lee91]** E. Lee: Hardware Overview of RAID 2 , UC Berkeley RAID Retreat, Lake Tahoe, January 1991.

102. **[Lega92]** Legacy Storage Systems: Back to Basics — A Disk Array Technology Primer, Computer Technology Review, October 1992, p. 42.

103. **[LeKa91]** E.K. Lee and R.H. Katz: Performance Consequences of Parity Placement in Disk Arrays, Proc. Fourth Int'l Conf. on Architectural Support for Programming Langs. and OS, pp. 190–199, April 1991.

104. **[LeKa93]** E.K. Lee and R.H. Katz: An Analytic Performance Model of Disk Arrays, Proc. SIGMETRICS '93, pp. 98–109, 1993.

105. **[LERK91]** Lee, Edward K., Randy H. Katz: Performance Consequence of Parity Placement in Disk Arrays, Fourth International Conference on Arhitectural Support for Programming Languages an Operating Systems (ASPLOS IV), Palo Alto CA, April 1991.

106. **[LiKB87]** M. Livny, S. Khoshafian, and H. Boral: Multi-Disk Management Algorithm, Proc. SIGMETRICS Conf., pp. 69–77, May 1987.

107. **[LiMP93]** J.J. Lim, J. Menon, D. Palmer: A Distributed Development Environment for Embedded Software. SP&E 23(11): pp. 1235–1248.

108. **[LoMa92]** R. W-M. Lo and N.S. Matloff: A Probabilistic Limit on the Virtual Size of Replicated Disk Systems , IEEE Transactions on Knowledge and Data Engineering, February 1992.

109. [LoMo94] D.D.E. Long, B.R. Montague: Swift/RAID: A Distributed RAID System. Computing Systems 7(3): pp. 333–359, 1994.

110. **[Lync72]** W.C. Lynch: Do Disk Arms Move? Performance Evaluation Rev., vol. 1, pp. 3–16, December 1972.

111. **[MaMc88]** A. Mahmood, E.J. McCluskey: Concurrent Error Detection Using Watchdog Processors — A Survey. IEEE Transaction on Computers, February 1988, vol. 37, pp. 160–174.

112. **[Mank91]** P.S. Manka: Direct Acces Storage Device with Independently Stored Parity, US Patent Number 0 469 924 A2, August 2, 1991.

113. **[Marc93]** R. Mărculescu: Tehnici de bază în proiectarea pentru testabilitate. Tempus Postgraduate School of Computer Aided Electrical Engineering. Editor Daniel Iona, Bucureşti, 1993.

114. **[Mass97]** P. Massglia: The RAIDbook, A Storage Technology Handbook, Sixth Edition, RAID Advisory Board, February 1997.

115. **[Matt91]** R.L. Mattson: Method for Balancing the Frequency of DASD Array Access When Operating in Degraded Mode, European Patent Application, Publication Number 0 469 924 a2, August 2, 1991.

116. **[Mccl90]** E. McCluskey: Design Techniques for Testable Embedded Error Checkers. IEEE Computer, July 1990, pp. 84–88.

117. **[Mead89]** W.E. Meador: Disk Arrays, CompCon, February 1989.

118. **[MeCo93]** J. Menon, J. Cortney: The Architecture of a Fault-Tolerant Cached RAID Controller. ISCA 1993: pp. 76-86.

119. **[MeKa89]** J. Menon and J. Kasson: Methods for Improved Update Performance of Disk Arrays, Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, 1991 (see also IBM Research Report RJ 6928, July 1989).

120. **[MeKa90]** J. Menon and J. Kasson: Methods for Improved Update Performance of Disk Arrays, Technical Report RJ 6928 (66034), IBM Almaden Research Center, Nov. 1990.

121. **[MeMa91]** J. Menon and D. Mattson: Performance of Disk Arrays in Transaction Processing Environments, CMG Transaction, Fall 1991.

122. **[MeMa92a]** J. Menon, D. Mattson: Comparison of Sparing Alternatives for Disk Arrays. ISCA 1992: pp. 318–329.

123. **[MeMa92b]** Performance of Disk Arrays in Transaction Processing Environments. ICDS 1992: pp. 302–309.

124. **[MeMa92c]** J. Menon and R. Mattson: Distributed Sparing in Disk Arrays, CompCon, February 1992.

125. **[MeMa92d]** J. Menon and D. Mattson: Comparison of Sparing Alternative for Disk Arrays, Proceedings of the International Symposium on Computer Architecture, May 1992.

126. **[MeMN91]** J. Menon, D. Mattson, and S. Ng, Performance of Disk Arrays in Transaction Processing Environments, Technical Report RJ 8230 (75424), IBM Almaden Research Center, July 1991.

127. **[Meno94]** J. Menon: Performance of RAID5 Disk Arrays with Read and Write Caching. Distributed and Parallel Database 2(3): pp. 261–293, 1994.

128. **[MeRK93]** J. Menon, J. Roche, and O.J. Kasson, Floating Parity and Data Disk Arrays, J. Parallel and Distributed Computing, vol. 17, pp. 129–139, 1993.

129. **[MeRW95]** J. Menon, Jeff Riegel, James C. Wyllie: Algorithms for Software and Low-Cost Hardware RAIDs. COMPCON 1995: pp. 411– 418.

130. **[MeTr97]** J. Menon, K. Treiber: Daisy: Virtual-Disk Hierarchical Storage Manager. SIGMETRICS Performance Evaluation Review 25(3): pp. 37–44, 1994.

131. **[MeYu91]** A. Merchant and P.S.Yu: Modeling and Comparisons of Striping Strategies in Data Replication Architectures, IBM Research Report RC 19960, 1991.

132. **[MeYu91]** A. Merchant and P.S. Yu: Modeling and Comparisons of Striping Strategies in Data Replication Architectures, IBM Technical Report, T.J. Watson Research Center, Sept. 1991.

133. **[MeYu92]** A. Merchant and P.S. Yu, Design and Modelling of Clustered RAID, Proc. 22nd FTCS, pp. 140–149, Boston, July 1992.

134. **[MeYu95]** A. Merchant and P.S. Yu, Analytic Modeling and Comparisons of Striping Strategies for Replicated Disk Arrays, IEEE Trans. on Computers, vol. 44, No. 3, pp. 419–433, March 1995.

135. **[MeYu96]** A. Merchant, P.S.Yu: Analytic Modeling of Clustered RAID with Mapping Based on Nearly Random Permutation. IEEE Transaction on Computers 45(3): pp. 367–373, 1996.

136. **[Micr91]** Micropolis Corp.: Drive Crush Terror Boosts RAID Market, Computer Tehnology Review, May 1991.

137. **[MiGr91]** C.A. Milligan and J.M. Graziano: Logical Track Write Scheduling System for a Parallel Disk Drive Array Data Storage Subsystem, International Patent Application, Publication Number WO 91/16711, October 31,1991.

138. **[MiMo96]** S.K. Mishra, P. Mohapatra: Performance Study of RAID 5 Disk Arrays with Data and Parity Cache. ICPP, vol. 1, 1996: pp. 222–229.

139. **[MoFS91]** A.N. Mourad, W.K. Fuchs and D.G. Saab: Performance Evaluation of Redundant Disk Array Support for Transaction Recovery, Coordinated Science Laboratory, University of Illinois, Urbana, Nov. 11, 1991.

140. **[MoKi94]** K. Mogi, M. Kitsuregawa: Dynamic Parity Stripe Reorganization for RAID 5 Disk Arrays. PDIS 1994: pp. 17–26.

141. **[MoKi96]** K. Mogi, M. Kitsuregawa: Hot Mirroring: A Study to Hide Parity Upgrade Penalty and Degradations During Rebuilts or RAID 5. SIGMOD Conference 1996: pp. 183–194.

142. **[Moru94]** C. Morun: Stadiul actual al implementării metodelor de fiabilizare la transmiterea şi stocarea datelor. Referat de doctorat, ianuarie 1994.

143. **[Moru94]** C. Morun: Metode de protecţie la transmiterea informaţiei. Referat de doctorat, iunie 1994.

144. **[Moru94]** C. Morun: Implementarea in reţele de calculatoare a protecţiei informaţiei. Referat de doctorat, iunie 1994.

145. **[Moru98a]** C. Morun: RAIC, Conferinţa CERF Banat '98, noiembrie 1998.

146. **[Moru98b]** C.Morun: RAID Level 7, SAFECOMP '99, 18th Int'l. Conf. on Computer Safety, Reliability & Security, Toulouse, France — lucrare trimisă spre recenzie.

147. **[Moru98c]** C.Morun : RAIC and Fault Tolerance, SAFECOMP '99, 18th Int'l. Conf. on Computer Safety, Reliability & Security, Toulouse, France — lucrare trimisă spre recenzie.

148. **[Moru98d]** C. Morun: RAID 7 vs RAID 5 with Spare Disks, Seventh Ann.Working Conf. on Information Security Management & Small Systems Security, Amsterdam, the Netherlands — lucrare trimisă spre recenzie.

149. **[Moru98e]** C.Morun: Striping Strategies for RAIC, SAFECOMP '99, 18th Int'l. Conf. on Computer Safety, Reliability & Security, Toulouse, France — lucrare trimisă spre recenzie.

150. **[Moru98f]** C. Morun: An algoritm for "nFT RAIC", COMPSAC '99, 23rd Int'l Computer Software & Applications Conf., Phoenix AZ – lucrare trimisă spre recenzie.

151. **[MoSt97a]** C. Morun, A. Stanciu. RAID with Multiple Fault Tolerance. Buletinul Ştiinţific al Universităţii „Politehnica“ din Timişoara, România, tomul 42 (56), 1997, pp. 131–136.

152. **[MoSt97b]** C. Morun, A. Stanciu. Better Results with RAID. Buletinul Ştiinţific al Universităţii „Politehnica“ din Timişoara, România, tomul 42 (56), 1997, pp. 137–142.

153. **[MuLu90]** R.R. Muntz and J.C. Lui: Performance Analysis of Disk Arrays Under Failure, Proc. 16th VLDB Conf., pp. 162–173, Brisbane, Australia, 1990.

154. **[NaKa88]** T. Nanya, T. Kawamura: Error Secure/Propagating Concept and its Application to the Design of Strongly Fault Secure Processors. IEEE Transaction on Computers, January 1988, vol. 37, pp. 14–24.

155. **[NCR91a]** NCR Corporation: NCR Disk Arrays: Solutions For Your Business Environment NCR publication, copyright 1991.

156. **[NCR91b]** NCR corporation: What Are Disk Arrays? NCR Publication, copyright 1990, 1991.

157. **[NCR91c]** NCR Corp.: Disk Arrays Diving Into The Computer Mainstream, Computer Tehnology Review, October, 1991, p. 12.

158. **[NCR93a]** NCR Corporation: Can Fault Resilience Protect RAIDs?, Computer Technology Review, February, 1993, p. 28.

159. **[NeWO88]** M. Nelson, B. Welch, and J. Ousterhout: Caching in the Sprite Network File System, ACM Trans. on Computer Systems, vol. 6, No. 1, pp. 134–154, Feb. 1988.

160. **[Ng87]** S. Ng: Design Alternatives for Disk Duplexing , IBM Research Report RJ 5481, January 30, 1987.

161. **[Ng89]** S. Ng: Pitfalls in Deigning Disk Arrays, CompCon, February 1989.

162. **[Ng89]** S. Ng: Some Design Issues of Disk Array, Proc. IEEE Spring ComCon, pp. 137–142, 1989.

163. **[NgLS88]** S. Ng, D. Lang, and R. Selinger: Trade-Offs Between Devices and Paths in Achieving Disk Interleaving, Proc. 15th Int'l Symp. Computer Architecture, pp. 196–201, Hawaii, May 1988.

164. **[NgMa94]** S. Ng and R.L. Mattson: Uniform Parity Group Distribution in Disk Arrays with Multiple Failures, IEEE Trans. on Computers, vol. 43, No. 4, pp. 501–506, April 1994.

165. **[OgFl90]** M. Ogata and M. Flynn: A Queueing Analysis for Disk Array Systems, Technical Report CSL-TR-90-443, Stanford Univ., August 1990.

166. **[Ohre90]** E. Ohrenstein: Supercomputers Seek High Throughput and Expandable Storage, Computer Technology Review, Spring 1990.

167. **[Olso89]** T.M. Olson: Disk Array Performance in a Random 10 Environment, Computer Architecture News, September 1989.

168. **[OsDo91]** J. Osterhout and F. Douglis: Beating the I/O Bottleneck: A Case for Log-Structured File Systems, CMG Transactions, Fall 1991.

169. **[Ouch78]** N.K. Ouchi: System for Recovering Data in Failed Memory Unit, US Patent Number 4 092 732, May 30, 1978.

170. **[OuDo89]** J.K. Ousterhout and F. Douglis: Beating the I/O Bottleneck: A Case for Log-Structured File Systems, Operating System Rev., vol. 23, No. 1, pp. 11–28, Jan. 1989.

171. [PaBa86] A. Park and K. Balasubramanian: Providing Fault Tolerance in Parallel Secondary Storage Systems, Technical Report CS-TR-057-86, Princeton University, November 1986.

172. **[PAGK88]** D.A. Patterson, G. Gibson, and R.H. Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID), Proc. ACM SIGMOD Conf., pp. 109–116, July 1988.

173. **[PaGK89]** D.A. Patterson, G.Gibson and R. H.Katz: Introduction to Redundant Arrays of Inexpensive Disks (RAID), CompCOn, February 1989.

174. **[PCGK89]** D.A. Patterson, P. Chen, G. Gibson, and R.H. Katz: Introduction to Redundant Arrays of Inexpensive Disks (RAID), Proc. IEEE Spring ComCon, pp. 112–117, 1989.

175. **[PDGG88]** Patterson, David A., Garth A. Gibson, Randy H. Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID) Proceedings of the 1988 ACM SIGMOD Conference on Management of Data, Chicago IL, June 1988, pp. 109–116.

176. **[Plan97]** J.S. Plank: A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems. SP&E 27(9): pp. 995–1012, 1997.

177. **[Prad86]** D.K. Pradhan. Fault Tolerant Computing: Theory and Techniques. Vol. I & II, Prentice-Hall, 1986.

178. **[Quan93]** Quantum Corporation: SCSI Array Features Can Cost Integrators, Technology Review, July 1993, p. 14.

179. **[ReBa89]** A.L.N. Reddy and P. Banerjee: An Evaluation of Multiple-Disk I/O Systems, IEEE Trans. Computers, vol. 38, No. 12, pp. 1680–1690, Dec. 1989.

180. [**ReBa91**a] A.L.N. Reddy and P. Banerjee: Gracefully Degradable Disk Arrays, Proc. 21st Int'l Symp. on Fault-Tolerant Computing, pp. 401–408, Montreal, June 1991.

181. **[Redd91a]** A.L.N. Reddy: Reads and Writes: When I/O's aren't quite the same, IBM Research Report RJ 8033, March 1991.

182. **[Redd91b]** A.L.N. Reddy: Reads and Writes: When I/O's Aren't Quite the Same, Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, 1991.

183. **[RiMe96]** J. Riegel, J. Menon: Performance of Recovery Time Improvement Algorithms for Software RAIDs. PDIS 1996.

184. **[RoFr94]** J.T. Robinson and P.A. Franaszek: Analysis of Reorganization Overhead in Log-Structured File Systems, Proc. 10th Int'l Conf. Data Eng., pp. 102–110, 1994.

185. **[RoOs91a]** M. Rosenblum and J.K. Osterhout: The LFS Storage Manager, CMG Transactions, Fallus 1991.

186. **[RoOs91b]** M. Rosenblum and J. Osterout: The Design and Implementation of a Log Structured File System, Proceedings of the Thirteenth ACM Symposium on Operating System Principless, October 1991.

187. **[RoOu90]** M. Rosenblum and J.K. Ousterhout: The LFS Storage Manager, Proc. Summer '90 USENX Technical Conf., pp. 315–324, Anaheim, Calif., June 1990.

188. **[RoOu91]** M. Rosenblum and J.K. Ousterhout, The Design and Implemenation of a Log-Structured Fils System, Draft, March 1991.

189. **[RoOu92]** M. Rosenblum and J.K. Ousterhout, The Design and Implementation of a Log-Structured File System, ACM Trans. on Computer Systems, vol. 10, No. 1, pp. 26–52, Feb. 1992.

190. **[RuOk94]** T. Ruwart and M. O'Keefe: Performance Characteristics of a 100 Megabyte/Second Disk Array, presented in Storage & Interface '94, Jan. 1994.

191. **[SaGM86]** K. Salem and H. Garcia-Molina: Disk Striping, Proceedings of the Second International Conference on Data Engineering, 1986.

192. [**SaMo86**] K. Salem and H. Garcia-Molina: Disk Striping, Proc. IEEE Data Eng. Conf., pp. 336–342, Los Angeles, Feb. 1986.

193. **[SaTN86]** M. Saito, T. Takeda, M. Nunotani: An Evaluation of Error Correcting Codes for Optical Disks: Paper of Technical Group IT86-48, IECE Japan, 1986.

194. [ScBu92] T.J.E. Scwarz, W.A. Burkhard: RAID Organization and Performance. ICDCS 1992: pp. 318– 325.

195. **[Schu88]** M.E. Schultze: Consideration in the Design of a RAID Prototype, Tehnical Report UCB/CSD 88/448, UC Berkeley, August 1988.

196. [**SGKP89**] M. Schulze, G. Gibson, R. Katz, and D. Patterson: How Reliable is a RAID? Proc. IEEE Spring ComCon Conf., pp. 118–123, San Francisco, Feb. 1989.

197. **[SHCG94]** D. Stodolsky, M. Holland, W.V. Courtright, and G.A. Gibson: Parity-Logging Disk Arrays, ACM Trans. on Computer Systems, vol. 12, No. 3, pp. 206–235, Aug. 1994.

198. **[Staf91]** Staff: Arrays More For Bandwidth Then Capacity, Computer Tehnology Review, October, 1991, p. 12.

199. **[Staf92a]** Staff: RAID Breaks Out Of Level 5 Straightjacket, Computer Technology Review, March, 1992, p. 1.

200. **[Staf92b]** Staff: RAID Board To Play The Number Game, Computer Technology Review, November, 1992, p. 41.

201. **[Staf92c]** Staff: Raid Becoming The People's Choice, Computer Technology Review, August, 1992, p. 36.

202. **[Staf93a]** Staff: No Cop-Outs on Backup Whith RAID, Computer Technology Review, February, 1993, p. 1.

203. **[Staf93b]** Staff: Is SCSI Out of Sync With RAID?, Computer Technology Review, March, 1993, p. 1.

204. **[Staf93c]** Staff: Controllers Basic to RAID Success , Computer Technology Review, May 1993, p. 35.

205. **[Staf93d]** Staff: Arrays: Hardware of Software RAID, Computer Technology Review, July 1993, p. 26.

206. **[Staf93e]** Staff: How to Select Between RAID and Solid State, Computer Technology Review, September, 1993, p. 41.

207. **[Staf93f]** Staff: RAID Fights Fog for Channel , Computer Technology Review, October, 1993, p. 1.

208. **[Staf93g]** Staff: Raid 6 Adding Another Level of Confusion, Computer Technology Review, November, 1993, p. 1.

209. **[Staf94a]** Staff: RAID Has To Get Its Numbers Straight And Fast, Technology Review, January 1994, p. 1.

210. **[Staf94b]** Staff: Certification Coming, Technology Review, January 1994, p. 1.

211. **[Ston90]** M. Stonebreaker and G.A. Schloss: Distributed RAID.

212. **[Stor91]** Storage Concepts, Is RAID The Disk Strategy For The Masses?, Computer Tehnology Review, May 1991, p. 23.

213. **[Stor92]** Storage Concepts: RAID5 Is Less Than Expected, Computer Tehnology Review, March, 1992, p. 12.

214. **[Stor94]** Storage Dimensions: Which RAID Meets the Application?, Technology Review, February 1994, p. 37.

215. **[StSc90]** M. Stonebreaker and G.A. Schloss: Distributed RAID — a New Multiple Copy Algorithm, Proceedings of the 6th International Conference on Data Engineering, February 1990.

216. **[StSc90]** M. Stonebraker and G.A. Schloss: Distributed RAID — A New Multiple Copy Algorithm, Proc. Sixth Int'l Conf. Data Eng., pp. 430–437, Feb. 1990.

217. **[Tech93]** Technology Forums, Ltd.: Is RAID Five Levels of Confusion?, Technology Review, February 1993, p. 37.

218. **[ThMe94]** A. Thomasian, J. Menon: Performance Analysis of RAID 5 Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation. ICDE 1994: pp. 111–119.

219. [ThMe97] A. Thomasian, J. Menon: RAID 5 Performance with Distributing Sparing. IEEE Transaction on Parralel and Distributed Systems 8 (6): pp. 640–657, 1997.

220. **[Thor73]** E. Thorp: Nonrandom Shuffling with Applications to the Game of Faro, JASA, vol 68, pp. 842-847, 1973.

221. **[Toy88]** W.N. Toy: Fault Tolerant Computing, 1988.

222. **[TPBG93]** F.A.Tobagi, J. Pang, R. Baird, M. Gang: Streaming RAID: A Disk Array Management System for Video Files. ACM Multimedia, 1993: pp. 393–400.

223. **[TsLe97]** W.J. Tsai, S.Y. Lee: Multi-Partion RAID: A New Method for Improving Performance of Disk Arrays under Failure. The Computer Journal 40 (1): pp. 30–42, 1997.

224. [**Vasu88**] A. Vasudeva: A Case for Disk Array Storage System, Proceedings of Reliability Conference, Santa Clara CA, 1988.

225. **[VINH80]** L.B. Vries, K.A. Imink, J.G. Nibor, S. Abe: The Compact Disc Digital Audio System — Modulation and Error Correction: Proceedings of the Sixty-Seventh AES Convention (October 1980), No. 1674 (H-8).

226. **[Vlad82]** M. Vlăduţiu: Contribuţii la creşterea coeficientului de disponibilitate al echipamentelor automate de prelucrare a informaţiei prin testare neconvenţională. Teză de doctorat, Bucureşti, 1982.

227. **[VlCr89]** M. Vlăduţiu, M. Crişan: Tehnica testării echipamentelor automate de prelucrare a datelor. Editura Facla, Timişoara, 1989.

228. **[VlPe94a]** M. Vlăduţiu, N.S. Petrakis.: About New Signature Generation Techniques for Binary Sequences. International Conference on Technical Informatics. Proceedings, vol. 5, Timişoara, 1994, pp. 11–15.

229. **[VlPe94b]** M. Vlăduţiu, N.S. Petrakis: Adapted Combinational Array for Exact Binary Division with Signed Operands. International Conference on Technical Informatics. Proceedings, vol. 5, Timişoara, 1994, pp. 1–10.

230. **[VTPB94]** M. Vlăduţiu, D. Todincă, N.S. Petrakis, I. Brînzaş: The Principle of a Fuzzy Processor with Parity Check. Buletinul Ştiinţific al U.T. Timişoara, 1994, tomul 39 (53), seria Automatizări şi Calculatoare.

231. **[WeZS91]** G. Weikum, P. Zabback and P. Scheuermann: Dynamic File Allocation in Disk Arrays, Proccedings of the 1991 ACM SIGMOD, May 1991.

232. **[WGSS95]** J. Wilkes, R.A. Golding, C. Staelin, T. Sullivan: The HP AutoRAID Hierarchical Storage Syst. SOSP 1995: pp. 96–108.

233. [WGSS96] J.Wilkes, R.A. Golding, C. Staelin, T. Sullivan: The HP AutoRAID Hierarchical Storage Syst. TOCS 14(1): pp. 108–136, 1996.

234. **[Yama86]** A. Yamagishi: Encoder and Decoder LSIs for BCH Code and RS Code: Proceedings of 1986 Workshop on Coding Theory and Its Applications WCTA86-2 (August 1986).

235. **[Yarm90]** V.N. Yarmolik. Fault Diagnostics of Digital Circuits. New York Wiley, 1990.