Universitatea POLITEHNICA din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Algoritmi concurenţi de acces la disc în timp real

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de

Inginer în domeniul Calculatoarelor şi Tehnologiei Informaţiei

programul de studii de licenţă Ingineria Informaţiei

Conducător ştiinţific

Conf. Dr. Ing. Ştefan STĂNCESCU

Absolvent

Dragoş Ciprian ENOIU

2014

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

Aprobat Director de Departament :

Prof. Dr. Ing. Sever PAŞCA

TEMA PROIECTULUI DE DIPLOMĂ
a studentului ENOIU D. Dragoş Ciprian, grupa 442A

1. Titlul temei: **Algoritmi concurenţi de acces la disc in timp real**

2. Contribuţia practică, originală a studentului va consta în:

Se vor *simula algoritmi de acces in timp real la disc pe sistemul de operare Windows / Linux, cu ajutorul tool-ul DiskSim, pentru diferite scenarii de lucru ale unor aplicatii. Se vor interpreta rezultatele simulării şi se vor compara performantele. Se va determina un criteriu si un algoritm optim corespunzator de acces la disk în timp real, pentru cazurile de concurenţă in accesul de date de pe disk din partea a două sau mai multe procese cu conditionari de limita de timp.*

3. Proiectul se bazează pe cunoştinţe dobândite în principal la următoarele discipline:
Structuri de Date si Algoritmi
Sisteme de Operare
Arhitectura Sistemelor de Calcul

4. Realizarea practică/ proiectul rămân în proprietatea: *UPB*

5. Proprietatea intelectuală asupra proiectului aparţine: *UPB*

6. Locul de desfăşurare a activităţii: *UPB/ETTI*

7. Data eliberării temei: 4 Decembrie 2013

CONDUCĂTOR LUCRARE:

Conf.Dr. Ing.Ştefan STANCESCU

STUDENT:

Dragoş Ciprian ENOIU

Anexa 5

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Algoritmi concurenţi de acces la disc în timp real*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Calculatoare şi Tehnologia Informaţiei*, programul de studii *Ingineria Informaţiei* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

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

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

Bucureşti, *26.06.2014*

Absolvent *Dragoş Ciprian ENOIU*

CUPRINS

Lista Acronimelor

FCFS	**First Come First Served (Primul venit, primul servit)**
SSTF	**Shortest Seek Time First (Cea mai rapidă căutare)**
SCAN	**Sau Elevator (Lift)**
SDF	**Shortest Distance First (Prima cea mai mică distanţă)**
C-SCAN	**Circular-SCAN**
C-LOOK	**Circular-LOOK (Căutare circulară)**
DS	**Deadline-Sensitive (Sensibil la termenul limită)**
SSEDO	**Shortest Seek Earliest Deadline by Order**
	(Cea mai rapidă cautare prin cel mai devreme termen limita dupa ordine)
SSEDV	**Shortest Seek Earliest Deadline by Value**
	(Cea mai rapidă cautare prin cel mai devreme termen limita dupa valuare)
CDS-SCAN	**Concurent Deadline Sensitive-SCAN**
HDD	**Hard Disk Drive**
RPM	**Revolutions Per Minute (rotaţii pe minut)**
RAID	**Redundant Array of Independent Disks**
LBA	**Logical Block Adressing**
NCQ	**Native Command Queuing**

Lista Figurilor

Introducere

Lucrarea işi propune simularea unor algoritmi de acces la disc cu ajutorul tool-ului Disksim.

Datorită avansării domeniului tehnologic întrun ritm alert, numeroşi algoritmi de acces au fost dezvoltaţi cu scopul de a ţine pasul cu cantitatea mare de date.

Concurenţa joacă un rol foarte important în gestiunea informaţiei pe disc, reuşind să crească performanţa sistemului.

Algoritmii de acces la disc pot fi împărţiţi în două categorii mari: Algoritmii ce nu prezintă restricţii în timp şi cei în timp real unde ratarea unui termen limită poate duce la o catastrofă în sistem.

Această lucrarea are ca obiectiv principal prezentarea în detaliu unor algoritmi concurenţi în timp real şi stabilirea unui criteriu de performanţa în funcţie de un scenariu prevăzut.

În primul capitol sunt trataţi algoritmii, cu scopul de a întelege cum funcţionează aceştia şi sa putem stabili un criteriu de performanţă atunci când două sau mai multe procese sunt concurente şi prezintă condiţie de limită de timp.

Cel de al doilea capitol face o scurtă prezentare asupra modului de gestionare şi funcţionare al NCQ-ului şi al unor caracteristici asupra HDD-urilor.

În al treilea capitol este prezentat simulatorul Disksim. Aşa cum am studiat în capitolul întâi, partea teoretică, putem programa simulatorul pentru a emula aceşti algoritmi concurenţi. Varianta utilizată a simulatorului Disksim este 4.0.

În ultimul capitol sunt prezentate rezultatele experimentale obţinute în urma simulării algoritmilor cu scopul de a stabili un algoritm optim în funcţie de scenariul folosit.

Lucrarea se încheie cu prezentarea concluziilor finale ale proiectului de diplomă.

1. Algoritmi de acces la disc

1.1 Algoritmi standard

Deoarece procesorul este mult mai rapid decât discul, sunt şanse foarte mari de suprapunere de cereri înainte ca discul sa le poată servi pe toate. Deoarece discurile au timp de acces neuniform, rearanjarea cererilor la disc poate reduce semnificativ latenţa discului. De exemplu, două cereri pe aceaşi pistă sunt mai uşor de procesat în secvenţă decât cereri ce se află pe piste diferite. Un sistem de operare inteligent, va încerca să reducă latenţa şi să crească transferul pe disc prin reordonarea cererilor la disc pentru o utilizare mai bună a discului folosind algoritmi diferiţi. Performanţa acestor algoritmi este deobicei comparată cu aşa numitul **planificator aleator** în care cererile sunt selectate întrun mod aleator din coada de cereri suprapuse.

Pentru a explica mai bine acest fenomen vom lua ca model, un disc cu 200 piste. Braţul de citire/scriere va începe la traseul cu numărul 50. Coada cu cereri va conţine următoarele piste la care se află o cerere : 95, 180, 34, 119, 11, 123, 62, 64.

1.1.1 First Come - First Served (FCFS)

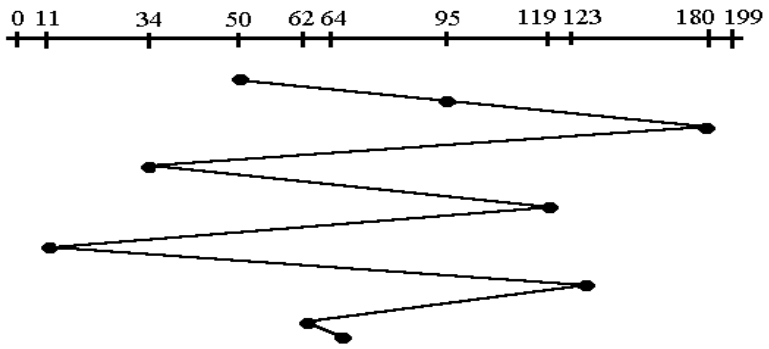

Fig. 1.1 **Algoritmul FCFS** [1]

Toate cererile ce sosesc sunt plasate în capătul cozii. Orice număr ce urmează din coadă va fi cel servit de către disc. Folosind acest algoritm nu vom avea cele mai bune rezultate. Pentru a determina distanţa parcursă de braţul de citire/scriere, vom număra traseele parcurse pentru a ajunge de la o cerere la alta. În acest caz, am trecut de la pista 50 la pista cu numărul 95, apoi la pista 180. De la pista cu numărul 50 până la 95, braţul de citire/scriere s-a mişcat 45 de piste. Dacă însumăm numărul de piste parcurse de la începerea algoritmului până la finalizarea lui, vom constata că avem un număr de 640 de piste parcurse în totalitate. Dezavantajul acestui algoritm este notat de faptul că

avem oscilaţii a braţului de citire/scriere de la traseul 50 la traseul 180 apoi se intoarce la traseul 11 urmat de numarul 123. După cum se observă acest algoritm are cele mai slabe performanţe pe care un disc îl poate folosi.

1.1.2 Shortest Seek Time First (SSTF)

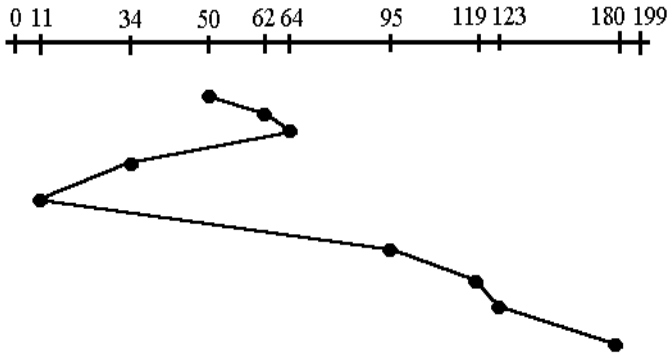

Fig. 1.2 **Algoritmul SSTF** [1]

În acest caz, cererile sunt servite în funcţie de cea mai apropiată distanţa de braţul de citire/scriere. Începând la 50, cea mai mică distanţă ar fi 62 în loc de 34 deoarece, aceasta se află la doar 12 piste distanţă de numărul 62 spre deosebire de 34 care se află la 16 piste distanţa de braţ. Procesul se repeta până când toate cererile sunt servite. De exemplu, în următorul caz, braţul se va muta de la traseul 62 la 64 în loc de 34 deoarece avem doar 2 piste de parcurs şi nu 18 dacă am fi mers în sensul opus. Deşi pare un algoritm mai bun decât cel prezentat precedent, cu o mişcare totală de 236 piste, nu este unul optimal. Motivul pentru care acest algoritm nu este unul optimal este acela că, în cazul în care avem multe cereri apropiate unele de altele, celelalte cereri mai îndepartate vor fi servite mult mai greu.

1.1.3 **Elevator (SCAN)**

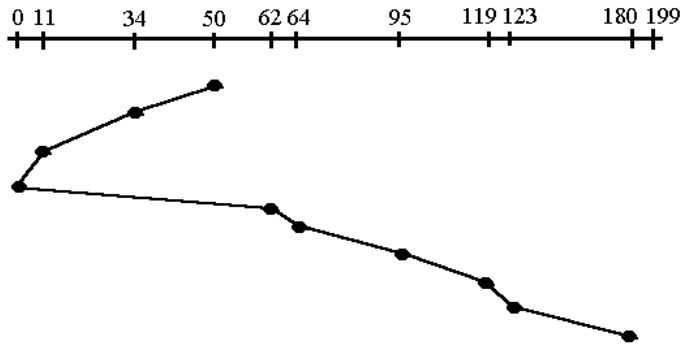

***Fig. 1.3* Algoritmul SCAN**

Această abordare a algoritmului este asemănătoare cu cea unui lift. Braţul scanează către cel mai apropiat capat al discului. Odată ajuns în capătul discului, braţul va scana în sensul opus spre celălalt capăt. Dacă o cerere soseşte imediat după ce braţul de citire/scriere a trecut, aceasta nu va fi servită până când se va afla pe direcţia de scanare. În cazul de faţa, braţul a parcurs un total de 230 de piste. Întradevar este un algoritm mai eficient decât precedentul dar nu şi cel mai bun.

1.1.4 Circular SCAN (C-SCAN)

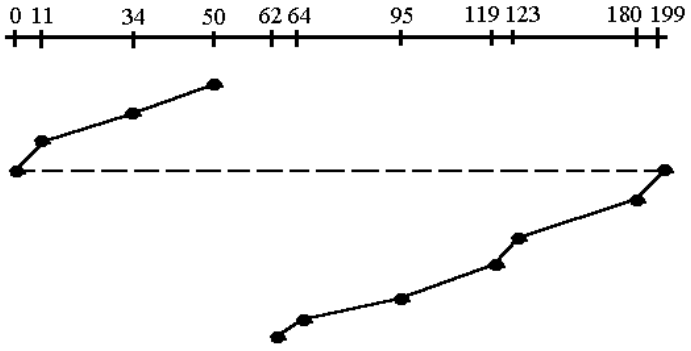

Fig. 1.4 Algoritmul C-SCAN [1]

Acest algoritm funcţionează în aceaşi manieră ca şi algoritmul anterior doar cu mici modificări. Braţul scanează în direcţia capătului de disc cel mai apropiat servind orice cerere pe care o întâlneşte. Odată ajuns la capăt, braţul de citire/scriere va sări în celălalt capăt al discului, miscându-se în aceaşi direcţie de scanare. De reţinut este faptul că nu vom considera saltul braţului dintrun capăt la celălalt ca fiind mişcare de scanare. Numărul total de piste parcurse este de 187, fiind un algoritm mai bun decât precedentul.

1.1.5 Circular LOOK (C-LOOK)

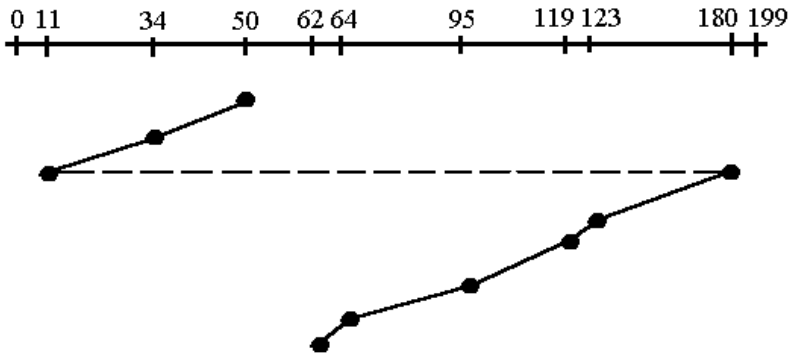

Fig. 1.5 **Algoritmul C-LOOK**

Acest algoritm este o versiune îmbunataţită a variantei C-SCAN. În cazul de faţă scanarea nu va trece de ultima cerere pe direcţia în care braţul se mişcă. Ca şi algoritmul precedent, braţul va sări în celălalt capăt dar nu al discului, ci a celei mai îndepărtate cereri. Se poate observa un număr total de 187 de piste parcurse în cazul algoritmului C-SCAN, însă cu algoritmul C-LOOK, am reuşit sa reducem numărul total la doar 157 de piste.

Aşadar, am reuşit să eficientizăm o metodă de scanare de la un număr total de 640 de piste parcurse la doar un număr de 157 de piste.

1.2 Algoritmi de acces la disc în timp real

Pentru un set de sarcini, problema generală a unui planificator este de a ordona cererile în funcţie de constrângerile pe care le are o cererea respectivă. În mod normal, o cerere este caracterizată de timpul de execuţie, timpul limită, timpul de îndeplinire a cererii şi resursele necesare. Executia unei sarcini poate fi întrerupta sau nu, în funcţie de algoritmul ce planifică cererile. Peste un set de cereri, putem avea o relaţie de prioritate ce restrânge ordinea de execuţie. Mai exact, execuţia unei sarcini nu poate începe până când toate sarcinile precedente au fost îndeplinite. În mod uzual, resursele sistemului determină ordinea cererilor de execuţie.

Un algoritm în timp real ar trebui sa îndeplinească următoarele condiţii:

- Atingerea constrângerilor legate de timp al sistemului.
- Prevenirea accesului simultan la resursele şi dispozitivele partajate.
- Atingerea unui grad înalt de utilizare a sistemului în funţie de constrângerile legate de timp.
- Reducerea costului comutatoarelor de context cauzate de preemţiune.
- Reducerea costului de comunicaţie al sistemelor distribuite în timp real.

De regulă, scopul planificatorului este de a determina ordinea de execuţie a cererilor astfel încât toate să fie îndeplinite înainte de termenul limită.

1.2.1 Deadline-Sensitive

Acest algoritm în timp real sortează fiecare cerere în doua clase, cele de timp real şi cele de tip best effort. În funcţie de termenul limită şi de dimensiunea cererii, o funcţie start-deadline poate fi computată. Această funcţie are rolul de caracteriza o cerere, ce prezintă un termen limită, dacă respectiva cerere poate fi îndeplinită la timp, presupunând cel mai prost timp de servire. Clasa ce cuprinde cererile de tip best effort nu prezintă termene limită, aşadar putem folosi alţi algoritmi cum ar fi algoritmul SCAN.

Algoritmul DSSCAN planifică cererile în funcţie de cele două cozi. Una ordonată de algoritmul Deadline-Sensitive iar cealaltă ordonată de algoritmul SCAN. De fiecare dată când o cerere cu termen limită este trimisă de un utilizator, aceasta este adaugată atât în coada de termene limită cât şi cea de tip best effort. În schimb, cererile ce nu prezintă termene limită sunt alocate doar în coada de tip best effort.

Planificatorul DSSCAN va servi o cerere din coada SCAN daca nicio cerere din cealaltă coadă nu işi va rata termenul limită. În caz contrar, planificatorul va servi cererea de disc cu cel mai scurt termen limită, apoi va rearanja coada SCAN întrun mod specific. Mai exact, dat fie cele două cozi de cereri, algoritmul DSSCAN foloseşte următoarele reguli când ia o decizie de planificare:

- Alegerea unei cereri din capătul cozii cu termene limită după politica Earliest Deadline First (Primul şi cel mai Devreme Termen Limita)
- Alegerea primei cereri din coada SCAN.

- Dacă cele două cereri sunt diferite, planificatorul va încerca să preia o cerere din coada SCAN doar dacă nicio cerere din coada cu termene limită nu işi va rata termenul. În caz contrar, prima cerere va fi preluată din coada cu termen limită.
- Ştergerea cererii selectată din ambele cozi şi rearanjarea cozii SCAN (daca este necesar) pentru a începe servirea cererii alese.

$Current = ED_{Ndisk}$;

for ($i=N_{disk}$; $i \geq 1$; i--)

{

$SD_i = \min (Current, ED_i) – X_i$;

$Current = SD_i$;

În codul de mai sus, avem algoritmul de calculare a timpului limită a unei cereri.

Ultimul pas este necesar din următoarele motive; putem considera atunci când cererea aleasă ce trebuie planificată nu se află în capătul cozii SCAN, dacă cererea aleasă este în capătul cozii cu termene limită. Odată ce cererea a fost servită , planificatorul va încerca să servească următoarea cerere din coada SCAN de unde braţul de citire/scriere a rămas în urma servirii cererii cu timp limită. Acest rearanjament poate fi făcut în mod constant dacă cele două liste sunt legate între ele printro legătură bidirecţională.

Acest algoritm are o proprietate de a menţine sistemul de transfer a traseelor în mod dinamic în funcţie de cât de strâns este termenul limită a cererilor date către disc. În scenariul în care termenii limită a cererilor sunt mai largi, algoritmul DSSCAN are tendinţa de a se comporta ca algoritmul SCAN crescând performanţele. Totuşi atunci când resursele necesare pentru cererile în timp real sunt aproape de capacitatea maximă, DSSCAN se axează pe cererile din coada cu termeni limită, astfel limitând performanţele sistemului.

1.2.2 SSEDO

Vom prezenta aceşti doi algoritmi în timp real SSEDO, acronim pentru Shortest Seek and Earliest Deadline by Ordering (Cea mai scurtă căutare şi cel mai apropiat termen limită prin ordonare) şi SSEDV, aconim pentru Shortest Seek and Earliest Deadline by Value (Cea mai scurtă căutare şi cel mai apropiat termen limită după valuare) pentru un singur disc.

Presupunem următorii parametri:

r_i – va fi cererea i cu cel mai scurt termen limită;

d_i – distanţa dintre poziţia braţului de citire/scriere şi poziţia cererii r_i;

L_i – termenul limită absolut al cererii r_i

Cei doi algoritmi menţin o coadă sortată după termenul limită absolut L_i pentru fiecare cerere. O fereastră de dimensiune m este definită ca fiind primele m cereri din coadă; aşadar cererile cu termenul limita cel mai mic vor fi în fereastra m.

În continuare vom discuta despre fiecare algoritm în parte în urmatoarele paragrafe.

Vom comenta despre algoritmul SSEDO. În această instanţă, planificatorul va selecta una dintre cereri din coadă, ce se află în fereastra de dimensiune m pentru a o servi. Regula este de a atribui fiecărei cereri o greutate, w_i pentru cererea r_i în care avem $w_1 = 1 \leq w_2 \leq .. \leq w_m$ unde evident m este dimensiunea ferestrei şi regula de a alege cererea cu cea mai mică valuare a lui wi*di. În continuare ne vom referi la această valuare wi*di ca valuarea prioritară asociată cererii r_i. Dacă avem cereri cu aceaşi valuare prioritară, atunci cererea cu termenul limită cel mai mic va fi servită. De reţinut faptul că aceşti doi parametri sunt variabili deoarece aceştia depind de poziţia braţului de citire/scriere.

Idea de bază al acestui algoritm este faptul că vrem să oferim cererilor cu termenii limită mai mici, o prioritate mai mare ca ele sa fie servite în timp util. Acest lucru este posibil atribuind valori mai mici la greutaţilor cererilor. Pe de altă parte, când o cerere cu un termen limită foarte mare se întamplă să se afle foarte aproape de braţul de citire/scriere, aceasta poate fi servită dacă planificatorul modifică parametrul d_i mărind prioritatea şi astfel reducem timpul de mişcare a braţului. Acest caz poate fi întărit atunci când braţul se află pe aceasi pistă cu cererea. Deoarece nu se pierde timp pentru mutarea braţului, timpul de căutare a cererii va fi dat doar de timpul de servire a cererii respective. În concluzie, astfel de cereri vor avea prioritate mai mare faţă de restul. Sunt diferite metode de atribuire de greutaţi w_i iar ca model vom lua formula următoare:

$$w_i = \beta^{i-1} \text{ cu } \beta \geq 1 \quad \text{iar } i = 1, 2, 3, ..,m.$$

În care β este un parametru ajustabil de către planificator. De observat faptul că valoarea w_i atribuie priorităţi doar pentru a odona cererile cu termen limită şi nu valorilor absolute sau relative. În cazul excepţional, când avem $\beta = 1$, obţinem algoritmul SSTF iar dacă mărim dimensiunea ferestrei m, vom obţine varianta pură de SSTF. Un alt scenariu este dacă avem dimensiunea ferestrei m de 1, atunci algoritmul se comportă ca ED (Earliest Deadline). În practică dimensiunea ferestrei este de 3 sau 4 cereri.

1.2.3. SSEDV

În algoritmul descris mai sus SSEDO, planificatorul se foloseşte doar de ordinea cererilor cu termen limită dar nu se foloseşte de diferenţa de timp dintre termenii limită a unor cereri succesive întro fereastră.

Presupunem că avem doua cereri în fereastra de dimensiune m, cererea r_1 are un termen limită foarte mic iar cererea r_2 are un termen limită mare. Dacă cererea r_2 se află fizic aproape de poziţia braţului de citire/scriere, conform algoritmului SSEDO, aceasta va fi servită ceea ce ar duce la ratarea termenului limită a cererii r_1 . Totuşi dacă r_1 este programată prima, atunci ambele cereri vor fi satisfăcute. În celălalt caz extrem, dacă avem termenul limită a cererii r_2, identic cu cel al cererii r_1 iar distanţa d_2 mai mică decât d_1 dar în acelaşi timp mai mare decat d_1 / β, algoritmul SSEDO va

planifica r_1 să fie prioritară faţă de r_2 cu consecinţa de a pierde cererea r_2. În acest caz, deoarece oricum este şansa mare de a pierde o cerere, este o alegere rezonabilă de a servi o cerere mai apropiată (r_2).

Bazându-ne pe aceste considerente, ne aşteptăm ca unii algoritmi, mai eficienţi şi inteligenti, nu se vor folosi doar ordinea cererilor cu termen limită, ci şi după ordinea cererilor luate după valuarea lor. Acest lucru duce la următorul algoritm : asociem o valuare de prioritate a unei cereri r_1 de $\alpha*d_i + (1 - \alpha)*l_i$ apoi să alegem cererea cu valuarea minima.

l_i reprezintă timpul de viaţă a cererii r_i definit ca timpul dintre timpul curent şi termenul limită absolut L_i al cererii respective.

$\alpha \in [0, 1]$ şi este un parametru folosit de către planificator.

Din nou, dacă $\alpha = 1$, avem algoritmul SSTF şi pentru $\alpha = 0$ vom obţine algoritmul ED.

O caracteristică comună acestor doi algoritmi SSEDV şi SSEDO este faptul că în ambele cazuri se ia în considerare timpul limită şi timpul de servire a unei cereri.

1.3 Accesul concurent la disc

Accesul concurent din punct de vedere al discurilor este atunci când mai multe persoane sau procese doresc accesul la acelaşi disc. Acest lucru poate fi problematic deoarece dacă un client sau un proces, accesează fişierul, acesta va fi rescris. Dacă fişierul va fi rescris, celalalt proces poate accesa fişierul rescris şi nu pe cel original. Această problema se poate complica cu sisteme de fişiere datorită concurenţei la suprapunere de scrieri pe disc. Această problema are soluţia simplă de gestionată de sisteme de locking (zăvoare) .

Un alt caz de concurentă la disc poate fi acela atunci când doua sau mai multe procese doresc accesul la disk pentru a scrie sau citi fişiere dar pe piste diferite. Planificatorul trebuie sa ia decizia de a ordona coada, în funcţie de prioritate şi termenii limită a cererilor.

1.3.1 CDS-SCAN

După cum am obervat în paragrafele anterioare, algoritmul DS-SCAN satisface condiţiile de planificare în timp real, cu presupunerea că estimarea timpului de satisfacere a cererii în cel mai rău caz este fezabil. În continuare putem construi algoritmul şi arăta faptul că acesta poate oferi rezultate în timp real, chiar şi atunci când avem cereri concurente. Se presupune că discul nu va avea un timp de procesare serial mai mare decât timpul de procesare concurent.

În general pentru un algoritm în timp real, cum ar fi DS-SCAN sau CDS-SCAN în particular, este necesar ca estimările cazurilor celor mai nefavorabile să fie corecte, acest lucru însemnând ca o cerere nu va avea un timp de prelucrare mai mare decât în cazul cel mai defavorabil. Această prezumţie este evidentă, deoarece dacă timpul estimat este prea mic, cererea poate fi procesată de catre disc după termenul limită, însa timpul de procesare este mai lung, deci sistemul poate rata termenul limită.

O coadă este fezabilă dacă toate cererile din coada respectivă îşi pot îndeplini termenii limită. Testul pentru determinarea fezabilităţii unei cozi este acela ca timpul limită de îndeplinire a cererii să nu fie mai devreme decât timpul curent. În schimb, e uşor ca o coadă cu cereri în timp real ce are termenul de depunere în trecut, să nu poată oferi garanţia îndeplinirii cererii, ratând termenul limită impus.

De exemplu, dacă avem două cereri, una cu termenul limita de 4ms şi una de 5ms, fiecare cu un caz defavorabil de servire de 3ms. În acest caz, termenul de predare a cererii va fi de 2ms în viitor iar cealaltă cerere va avea termenul de predare de 1ms în trecut. Dacă trimitem imediat prima cerere către disc iar timpul de servire va fi cel mai defavorabil de 3ms, atunci cererea respectivă va fi îndeplinită în timp util. Dacă, la rândul ei, cea de a doua cerere va avea un timp de servire întrun caz nefavorabil, cererea işi va rata termenul limită cu 1 ms după termenul limită.

Totuşi, estimarile în cele mai defavorabile cazuri, prezintă cel mai prost scenariu, deci sunt şanse mari ca cele doua cereri să fie îndeplinite în timp util chiar dacă algoritmul nu poate garanta reuşita cererilor. În cazul de faţă, dacă ambele cereri au timpul de servire de 2ms, atunci ambele vor fi servite la timp.

Lemă: O cerere în timp real ce a fost trimisă către disc va fi completată înainte de termenul ei limită atât timp cât termenul limită nu este mai devreme de cel de servire unor cereri concurente. În caz contrar, cererea ce îşi va rata termenul limită şi a fost trimisă către disc sau a fost trimisă după termenul limită.

2. Native Command Queuing

2.1 Introducere

Accesul la datele de pe disc, poate avea un impact negativ asupra sistemului, atunci când vorbim de performanţele sale. Spre deosebire de alte componente electronice, un astfel de sistem, cum este un HDD, este un dispozitiv mai mult mecanic. Aceste HDD-uri întâmpină dificultaţi datorită inerţiei asupra componentelor mecanice ce pot limita viteza de acces la date. Performanţele mecanice pot fi îmbunătaţite la nivel fizic însă doar până întrun punct, deoarece putem ajunge în cazul consumului unui volum de resurse mai mare cu îmbunătaţiri minore. Totuşi, management-ul intern, poate fi gestionat de un sistem inteligent cu scopul de a mări performanţele fără a ne atinge de aspectul mecanic al discurilor. Discul însuşi poate accesa zone diferite specificate cum sunt adresele blocurilor (LBA) şi să îşi aleagă singur cea mai bună soluţie cu scopul de atinge performanţe maxime.

Native Command Queuing este un protocol de comandă în serialul ATA ce permite mai multe cereri deodată să fie servite în interiorul discului în acelaşi timp. Drivere ce suportă NCQ, au o coadă internă unde comenzile pot fi servite întrun mod dinamic sau rearanjate în funcţie de mecanismul de urmărire pentru cereri de acest gen. Deasemenea, NCQ prezintă un mecanism ce permite unui utilizator să lanseze mai multe comenzi către disc în timp ce discul caută informaţia altei cereri. Cu alte cuvinte, nu se pierde timp când braţul este în miscare.

Sistemele de operare, cum sunt Microsoft Windows şi Linux, profită din ce în ce mai mult de avantajul protocolului NCQ.

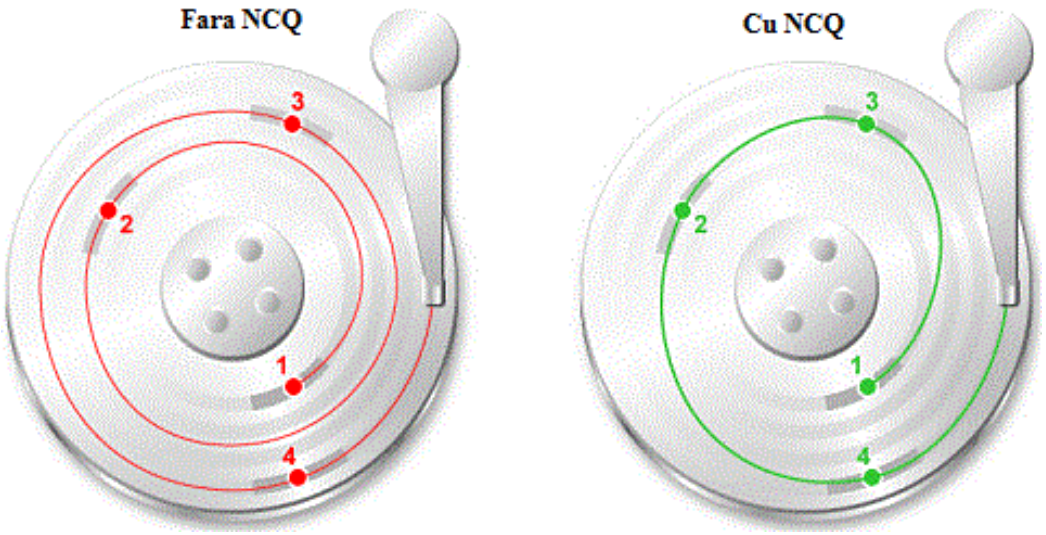

Fig. 2.1 **Efectul fără/cu NCQ**

Sursa: techreport.com

2.2 HDD

Hard disc-urile sunt dispozitive electromagnetice, deci prin urmare, sunt hibrizi ai electronicii şi mecanicii. Partea mecanică a dispozitivului se poate deteriora şi poate prezenta o limită atunci când este vorba de factorul de performanţă. Pentru a întelege procesul, în continure vom explica cum informaţia este scrisă pe disc:

Datele sunt scrise pe disc pe cercuri concentrice, numite piste, începând de la perimetrul exterior a discului cel mai jos pozitionat, discul 0 şi braţul de citire/scriere, bratul 0. Atunci când avem un cerc complet pe partea unui disc, dispozitivul începe să scrie cu braţul 1 dar tot pe pista 0. După o rotaţie, se trece la braţul 2 pe pista 0. Procesul se repetă pâna când ajungem la ultimul braţ, după care, se va trece la pista 1 pe braţul 0. Acest proces complex, produce cercuri concentrice în care datele vor fi din ce în ce mai apropiate de perimetrul interior.

O pistă particulara cu toate suprafeţele discurilor se numeşte cilindru. Aşadar datele sunt plasate pe disc întrun mod secvenţial în cilindri ce încep din perimetrul exterior.

O provocare majoră este faptul că, deobicei, procesele sau utilizatorii, au tendinţa să ceară accesul la date ce sunt împrăştiate pe un disc. Mişcarea mecanică necesară pentru a aduce braţul corect în poziţia finală nu este una trivială. Cu cât datele sunt mai împrăştiate, cu atât timpul de căutare va creşte.

Un algoritm cunoscut pentru minimizarea căutării şi a latenţei discului, poartă denumirea de Rotational Positioning Ordering (Ordonarea după poziţionarea rotaţională). Acest algoritm permite dispozitivului să îşi aleagă ordinea de execuţie a comenzilor către date în aşa manieră de a maximiza performanţa şi de a micşora timpul de acces. Timpul de aces reprezintă o sumă între timpul necesar de deplasare a braţului de citire/scriere în poziţia corectă şi timpul pierdut atunci când discul rotativ aduce informaţia sub braţul de citire/scriere. Ambele valori sunt de oridinul milisecundelor.

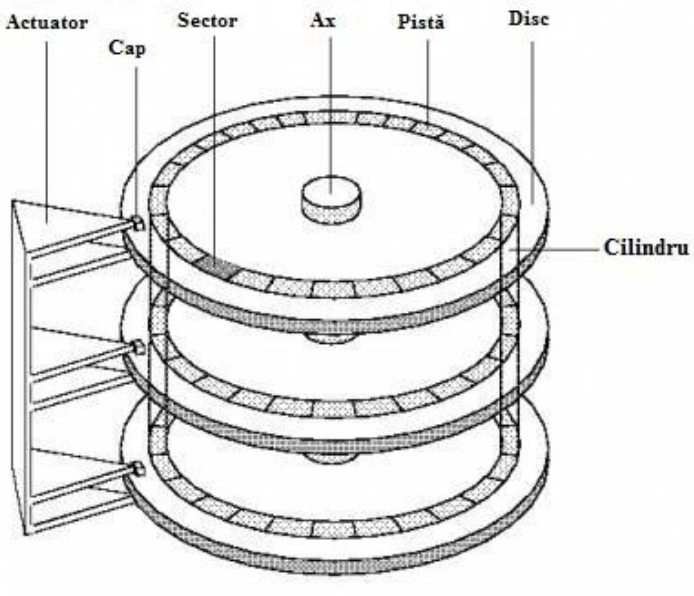

Fig. 2.2 **Componentele unui HDD**
Sursa: www.cs.ucla.edu

2.2 Optimizarea timpului de căutare

Latenţa datorită căutarii este produsă de timpul necesar braţului de citire/scriere să se poziţioneze pe pista potrivită ce conţine adresa logică a blocului (LBA) căutat. Pentru a îndeplini mai multe comenzi, discul trebuie să acceseze toate LBA-urile ţintă. Fără să avem o ordonare, discul va accesa LBA-urile în ordinea în care acestea sosesc. Totuşi dacă toate cererile vin spre disc întrun timp foarte scurt, discul poate satisface aceste cereri întrun mod optimal. Modul optimal este acela de a reduce timpul de căutare este dat de reducerea mişcării braţului de citire/scriere.

O analogie bună pe care o putem face, este cea unui lift. Dacă toate opririle sunt făcute în ordinea de apăsare a butoanelor, liftul va opera întrun mod foarte ineficient şi ar risipi cantităţi mari de energie şi timp plimbându-se dupa ordinea de apăsare. Deşi pare de necrezut, o sumedenie de algoritmi funcţionează după acest principiu al liftului. Prin ajutorul tehnologiei SATA, este posibilă rearanjarea comenzilor dintrun punct specific, chiar şi în mod dinamic, însemnând că în orice timp dat, comenzi noi pot fi adăugate în coadă. Noile comenzi sau cereri, vor fi introduse întrun proces sau stocate întrun buffer până când ii va veni rândul şi servită în timp util.

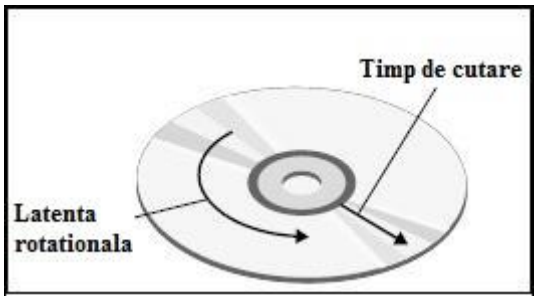

Fig. 2.3 **Timpul de căutare şi Latenţa rotaţională**

Sursa: cmcacademylko.blogspot.com

2.3 Latenţa Rotaţională

Latenţa rotaţională reprezintă timpl necesar ca un LBA să se rotească sub braţul de citire/scriere după ce acesta se află pe pista potrivită. În cel mai defavorabil caz, acest lucru lucru ar insemna că discul va trebui să risipească timpul şi să aştepte o rotire completă a discului ca după LBA-ul să fie poziţionat pentru citire/scriere. Această latenţă depinde de RPM, de exemplu un disc cu un RPM de 7200, va avea o latenţă în cel mai defavorabil caz de aproximativ 8.3ms, un disc cu o viteză de rotaţie de 5400, va vea 11.1ms.

Întârzierile sunt de ordinul milisecundelor dar pot afecta foarte mult peformanţa unui disc pe întreg ansamblu. Un exemplu elogvent este scenariul în care sistemul de operare utilizează multi-threading ce pot permite executarea quasi-simultană a unor fire de execuţie, toate ce au nevoie de date de pe acelaşi disc, aproape în mod simultan.

Marirea vitezei de rotaţie ar putea fi o soluţie pentru a reduce latenţa. Totuşi, mărind RPM-ul, axul poate fi suprasolicitat din punct de vedere fizic. În principiu, această latenţa este minimizată prin două metode. Prima metodă este de a rearanja cererile ce vin întrun mod, astfel încât, această

latenţă să fie minimizată. Această metodă este similară cu optimizarea liniară pentru reducerea timpului de căutare, doar că se ia în considerare poziţia braţului în determinarea servirii urmatoarei cereri. A doua metodă de optimizare, este de ne folosi de o metodă numită livrarea datelor out-of-order. Această metodă presupune că braţul nu trebuie să se aşeze fix peste LBA-ul dorit ci în zona lui. De îndată ce braţul începe a servi cererea în prejma blocului, acesta nu începe fix de la început ci va citi din poziţia în care a ajuns după care va completa cererea la capătul aceleaşi rotaţii.

Folosind această metoda, pentru cazul cel mai defavorabil, orice cerere, va fi completată fix întro rotaţie completă a unui disc. Fără acest mecanism, timpul total va fi aşteptarea LBA-ului apoi citirea zonei potrivite.

Cu toate aceste informaţii, ne putem pune întrebarea despre ce înseamnă timpul mediu de căutare. Este clar că acest timp mediu de căutare este media timpului de căutare în cel mai favorabil caz cu timpul de căutare atunci când avem cel mai defavorabil caz. Pentru a înţelege acest lucru trebuie să rezolvăm un model simplu pentru a ne putea pronunţa ce este exact timpul mediu de căutare.

Vom calcula distanţa medie parcusă de braţ conform următorului tabel:

Tabelul 2.1

Distanţa		Spre pista:					
		1	2	3	...	n-1	n
De la pista:	1	0	1	2	...	n-2	n-1
	2	1	0	1	...	n-3	n-2
	3	2	1	0	...	n-4	n-3
	...	...	...	...	...	...	...
	n-1	n-2	n-3	n-4	...	0	1
	n	n-1	n-2	n-3	...	1	0

Vom nota faptul că distanţa totală parcursă de capul de citire/scriere în n^2 cazuri este listată în tabelul de mai sus prin valoarea TD(n) ce va fi distanţa totală si distanţa medie AD(n).

$$AD(n) = \frac{TD(n)}{n^2} \text{ (Distribuţie uniformă de cereri)}$$

Putem observa o relaţie de recurenţă:

$$TD(1) = 0 \ (1)$$

$$TD(2) = TD(1) + 2(1)$$

$$TD(3) = TD(2) + 2(1+2+)$$

...

$$TD(n+1) = TD(n) + 2(1+2+...+n) = TD(n) + n(n+1) \ (2)$$

Vom presupune faptul ca funcţia TD(n) este una polinomială deci vom presupune următoarea formulă deoarece formula de mai sus are gradul 2 deci funcţia polinomială TD(n) nu poate avea o valoare mai mare decat 3:

$$TD(n) = a*n^3 + b*n^2 + c*n + d \quad (3)$$

$$TD(n+1) = a(n+1)^3 + b(n+1)^2 + c(n+1) + d = a(n^3 + 3n^2 + 3n + 1) + b(n^2+2n+1) + c(n+1) + d$$

$$TD(n+1) = an^3 + 3an^2 + 3an + a + bn^2 + 2bn + b + cn + c + d$$

$$TD(n+1) = an^3 + (3a+b)n^2 + (3a+ 2b + c)n + (a + b + c + d)$$

În continuare vom exprima membrul drept al formulei (2) folosindu-ne de formula (3):

$$TD(n) + n(n+1) = an^3 + bn^2 + cn + d + n(n+1) = an^3 + (b+1)n^2 + (c+1)n + d$$

După egalarea termenilor cu acelaşi argument vom avea următoarele ecuaţii:

$$3a + b = b + 1$$

$$3a + 2b + c = c + 1$$

$$a + b + c + d = d$$

Aşadar avem un sistem cu 3 ecuaţii dar 4 necunoscute. Ne va trebui încă o ecuaţie pentru rezolvare. Din relaţiile (1) si (3) obţinem

$$TD(1) = 0 = a*1^3 + b*1^2 + c*1 + d = a + b + c + d = 0$$

Din rezolvarea sistemului de ecuaţii cu cele 4 necunoscute:

$$a = \frac{1}{3}, \; b = 0, \; c = -\frac{1}{3}, \; d = 0.$$

Dacă vom înlocui în $TD(n) = \frac{n^3-n}{3}$ care o mai putem scrie sub forma:

$$TD(n) = \frac{n^3-n}{3} = \frac{(n-1)n(n+1)}{3} \quad (5)$$

$$AD(n) = \frac{TD(n)}{n^2}$$

Din formulele de mai sus rezultă egalitatea:

$$AD(n) = \frac{(n+1)(n-1)}{3n} = \frac{n^2-1}{3n} = \frac{(n-\frac{1}{n})}{3}$$

Pentru valori mari ale lui n, distanţa medie va fi : $AD(n) \approx \frac{n}{3}$

În cazul în care avem distribuţie neuniformă de cereri vom face următoarele presupuneri:

Vom presupune p0 probabilitatea ca o cerere să se afle pe diagonala cu 0, p1 probabilitatea de apariţie a cererii următoare la o distanţă de o pistă şi 1-p0-p1 probabilitatea resturilor de cereri.

R*p0 – cereri de date pentru aceaşi pistă

R*p1 – cereri de date pentru pista alăturată

R*(1-p0-p1) – cereri pentru oricare altă pistă

Distanţa medie parcusă pentru o cerere este:

0 –dacă avem aceaşi pistă

1 – dacă avem o pistă vecină

$\frac{TD(n)-(n-1)}{n^2-n-(n-1)}$ - pentru restul cererilor

Distanţa totală parcusă de braţ pentru R cereri va fi :

$$R*p0*0 + R*p1*1 + R*(1-p0-p1) *\frac{TD(n)-(n-1)}{n^2-n-(n-1)}$$

Dacă împărţim ecuaţia la R vom avea distanţa medie parcusă pentru R cereri :

$$AD(n) = p0*0 + p1*1 + (1-p0-p1) *\frac{TD(n)-(n-1)}{n^2-n-(n-1)}$$

Din această formulă vom înlocui pe TD(n) din formula (5):

$$AD(n,p0,p1) = p1 + \frac{(1 - p0 - p1)(n2 + 3 - n)}{3(n-1)} \quad (6)$$

Pentru valori mari ale lui n, formula (6) se poate simplifica:

$$AD(n,p0,p1) \approx (1- p0 - p1) - \frac{n}{3} \quad (7)$$

În următorul caz se va considera un disc cu n piste pe fiecare suprafaţă. Vom presupune o probabilitate ca următoarea cerere, pentru un transfer de date oarecare, va fi în jurul aceleaşi piste cu valoarea p_0 . Presupunem probabilitatea p_1, de apariţie cererii viitoare dar în care braţul de citire/scriere trebuie să se deplaseze către pista k+1, în care k < n. Deasemenea, vom presupune că timpul necesar deplasării braţului de citire/scriere de pe pista i, pe pista j va fi t_0 + |i-j|*t1, unde t_0 şi t_1 sunt constante. Se mai ştie faptul că t_0 este mai mare decât t_1. Cât va fi timpul mediu de căutare $AT(n,p_0,p_1,t_0,t_1)$? (AT- avrege time,timp mediu)

Vom aduce o valoare nouă în calcul, R ce va reprezenta numărul de cereri ce vor fi lansate către disc. Distanţa totală parcursă de braţ, va fi R* $AT(n,p_0,p_1,t_0,t_1)$. Întrun număr de $R*(1 - p_0)$ cereri, braţul nu va sta pe aceaşi pistă, astfel va trebui să începem contorizarea timpului de căutare. Timpul total pierdut pentru mişcarea mecanică a braţului în aceste R cereri poate fi împărţit în $R*(1 - p_0)$ porniri ale braţului însumând o valoare de aproximativ $R*(1 - p_0)*t_0$. Timpul pierdut datorită mişcării la o viteză constană pe toată distanţa este $R*AD(n, p_0,p_1)*t_1$, unde AD este distanţa medie. Aşadar timpul total pierdut pentru mişcarea braţului cu scopul servirii a R cereri va fi:

$$R*(1 - p_0)t_0 + R*AD(n, p_0,p_1)t_1$$

Dacă vrem să obtinem timpul mediu de căutare trebuie să impărţim toată ecuatia la R:

$$AT(n,p_0,p_1,t_0,t_1) = (1 - p_0)t_0 + AD(n, p_0,p_1)t_1$$

Din calcule matematice şi demonstraţiile anterioare s-a arătat faptul că:

$$AD(n,p_0,p_1) = p_1 + \frac{(1-p_0-p_1)(n^2 + n - 3)}{3(n-1)}$$

Aşadar timpul mediu $AT(n,p_0,p_1,t_0,t_1)$ va fi egal cu:

$$AT(n,p_0,p_1,t_0,t_1) = (1 - p_0)t_0 + p_1 + \frac{(1-p_0-p_1)(n^2 + n - 3)}{3(n-1)} * t_1$$

Această formulă poate fi simplificată atunci când avem un n foarte mare:

$$AT(n,p_0,p_1,t_0,t_1) = (1 - p_0)t_0 + (1-p_0-p_1)t_1 \frac{n}{3}$$

În această formulă aproximată, am lăsat constant termenul $(1 - p_0)*t_0$ deoarece t_0 este mult mai mare decât t_1. Aşadar acest termen va afecta mai tare valoarea rezultată dacă este n mai mare. În realitate, discurile calculatoarelor au valoarea lui n de orinul 100 sau 1000 în timp ce t_0 poate fi de 50 ori mai mare decât t_1.

2.4 Beneficiile NCQ-ului

Este clar faptul că este necesară o rearanjare a cererilor ce vin către disc pentru a reduce timpul mecanic şi pentru a îmbunataţi timpul pierdut pe latenţe. Deasemenea, este clar faptul că doar prin colectarea cererilor întro coadă putem economisi timp de lucru util. Algoritmi eficienţi de înregistrare, vor prelua date despre poziţia datelor ce trebuiesc servite şi vor optimiza timpul de căutare. Acest proces are denumirea de ‘rearanjare de comenzi bazate pe căutare şi optimizare rotaţională’ sau tagged command queuing. Un efect benefic, este faptul că braţul va fi supus mai puţin la tensiuni mecanice prelungind termenul de utilizare a discului. Serial ATA II ofera un protocol eficient implementat pentru TCG.

Native command queuing prezintă performanţe şi eficientizare ridicată prin rearanjarea comenzilor. În plus, sunt trei noi capabilitaţi ce vin odată cu protocolul Serial ATA pentru a îmbunătaţi performanţa NCQ-ului cum ar fi următoarele:

- *Race-Free Status Return Mechanism*

Acestă trăsătură permite comunicarea stării despre orice cerere la orice moment de timp dat. Nu există condiţia de ‘handshake’ necesară să fie lansată de un utilizator sau proces. Discul poate lansa comenzi de completare pentru o multitudine de cereri back-to-back sau cele în acelaşi timp.

- *Intrerupt Aggregation*

În mod general, de fiecare dată când discul completează o comandă, discul întrerupe utilizatorul. Cu cât avem mai multe întreruperi, cu cât putem observa un decalaj în performanţele sistemului. Totuşi dacâ avem NCQ, numărul mediu de întreruperi pe comandă poate fi chiar şi sub una. Dacă discul completează cereri multiple întrun timp scurt, întreruperile pot fi agregate. În acest caz, controler-ul trebuie doar să proceseze o întrerupere pentru o multitudine de comenzi.

- *First Party DMA*

NCQ are un mecanism de comandă, ce permite discului să opereze prin acces direct la memorie (DMA) pentru transferul de date fără ca sistemul de operare să intervină. Acest mecanism se numeşte First Party DMA. Discul va selecta contextul DMA-ului trimiţând un Setul FIS (Frame Information Structure) la controle-ul utilizatorului.

3. Simulatorul ‘Disksim’

Disksim este un simulator eficient, precis şi uşor de configurat, creat cu gândul de a beneficia cercetărilor în diferite aspecte ale arhitecturilor de stocare a subsistemelor. Acesta include modele ce simulează discuri, controlere intermediare, magistrale, drivere, cereri de acces, buffere şi organizarea datelor pe mulţimi de discuri cum sunt arhitecturile RAID. În particular, acest simulator poate reproduce scenarii complexe din viaţa reala cu o precizie foarte mare.

În continuare vom discuta despre cum putem să configurăm şi să folosim Disksim-ul. Acest simulator este valabil publicului în speranţa de a măsura performanţele unui sistem.

3.1 Descrierea parametrilor

Disksim necesită, în principiu, cinci parametri în linia de comandă dar poate accepta şi alţi parametri opţionali de suprascriere. Comanda de simulare a unui scenariu este următoarea:

disksim <parfile> <outfile> <tracetype> <tracefile> <synthgen>

- disksim reprezintă numele executabilului.
- partfile reprezintă fişierul cu parametrii de intrare
- outfile este numele fisierului în care se vor scrie datele de ieşire
- tracetype indică în ce tip de format se află fişierul cu datele de intrare.
- tracefile este numele fişierului de trace-uri care va fi folosit la intrare. Dacă datele de intrare sunt luat de la tastatură, se va specifica stdin pentru acest parametru.
- synthgen este parametrul ce specifică dacă generatorul de workload este activ sau nu. Orice valoare diferită de 0 va activa generatorul. În această lucrare vom folosi un workload personalizat.

Disksim poate fi uşor configurat prin fişierul de parametri pentru a simula o varietate de subsisteme. Acest simulator foloseşte fişierul libparam pentru a introduce fişierul de parametri. În acest fişier putem găsi o multitudine de valori ce pot fi schimbate în funcţie de necesităţi cum ar fi: topologii de sistem, blocuri, instanţieri, declarări, etc.

Un bloc reprezintă o multitudine de asignari de tipul ‘name = value’. Numele poate contine spaţii dar sunt sensibile la litere mari şi mici. Valorile folosite sunt deobicei în formă de numere zecimale întregi dar putem folosi, numere reale sau şiruri de caractere. Blocurile sunt delimitate de ‘[‘ şi ‘]’.

Libparam conţine o directivă numită şi *source* întrun mod similar cu #include, folosit de compilatoarele din C.

Organizarea parametrilor poate fi flexibilă deşi trebuie sa specificăm câteva blocuri ce le vom folosi. Modul de organizare al parametrilor vor respecta urmatorul format:

Numele Blocului	Numele Parametrului
Descrierea Parametrului	

Vom descrie câţiva parametri esenţiali ce au fost modificaţi pentru a ne atinge scopul acestei lucrari ştiinţifice:

disksim_iodriver	Use queuing în subsystem
Specifică faptul dacă driver-ul dispozitivului permite ca mai mult de o cerere să acceseze discul. În cazul nostru am folosit algoritmii FCFS, SSTF, SCAN şi SDF.	

disksim_iodriver	Scheduler
Acest parametru specifică driver-ului ce algoritm de acces la disc se va folosi pentru gestiunea datelor.	

disksim_ctlr	Maximum queue length
Specifică numărul maxim de cereri ce pot fi stocate de către controller-ul discului. În cazul nostru au fost folosit un număr maxim de cereri ce aşteaptă a fi servite de 5 cereri.	

disksim_ioqueue	Scheduling policy
Acest parametru specifică algoritmul primar de gestiune a datelor, ce va fi folosit atunci când se va servi următoarea cerere. Disksim permite simularea unui numar mare de algoritmi cu complexităţi diferite; de la cel mai simplu la cel mai complex. Algoritmii utilizaţi de acces la disc au fost următorii: FCFS, SSTF, SCAN, SDF.	

disksim_ioqueue	Timeout time/weight
Parametru ce specifică timpul limită, în milisecunde, folosit de unii algoritmi ce oferă priorităţi unor cereri ce au constrangeri legate de timp. Unii algoritmi folosesc acest parametru întrun mod aditiv pentru a da prioritate cererilor, altii întrun mod multiplicativ. Toate cererile generate au primit un timp limită distribuit după o gaussiană de medie 30 si dispersie 100	

disksim_ioqueue	Scheduling priority scheme
Valoare binara; 0 specifică faptul ca cererile cu prioritate nu vor fi stocate separat întro alta coadă. În caz contrar, cererile cu prioritate vor fi stocate intro coadă separată. Acest bit în cazul nostru a fost setat pe 1 pentru a utiliza această coada prioritară	

disksim_ioqueue	Priority scheduling
Daca se foloseşte o coadă separată pentru cererile cu prioritate, atunci trebuie să specificăm cu ce algoritm vom trata cererile din coada cu prioritaţi. În coada prioritară au fost folosiţi aceaşi algoritmi ca şi in cazul cozii de tip best effort. Aceşti algoritmi fiind: FCFS, SSTF, SCAN şi SDF.	

disksim_cachedev	Max request size
Specifică dimensiunea maximă a unei cereri ce poate fi servită de o memorie cache. Dacă o cerere este mai mare decât această valoare, atunci cererea nu poate fi stocată în cache cu riscul de a pierde cererea dacă avem acces concurent. Bufferul în acest caz a fost de aproximativ 50 adrese logice.	

Topologia Sistemului este una foarte importantă pentru simulator. Un exemplu de topologie pe care s-a lucrat şi au fost făcute simulările este:

```
topology disksim_iodriver driver0 [
        disksim_bus bus0 [
                disksim_ctlr ctlr0 [ [
                        disksim_disk disk0 [
                                ]
                        ]
                ]
```

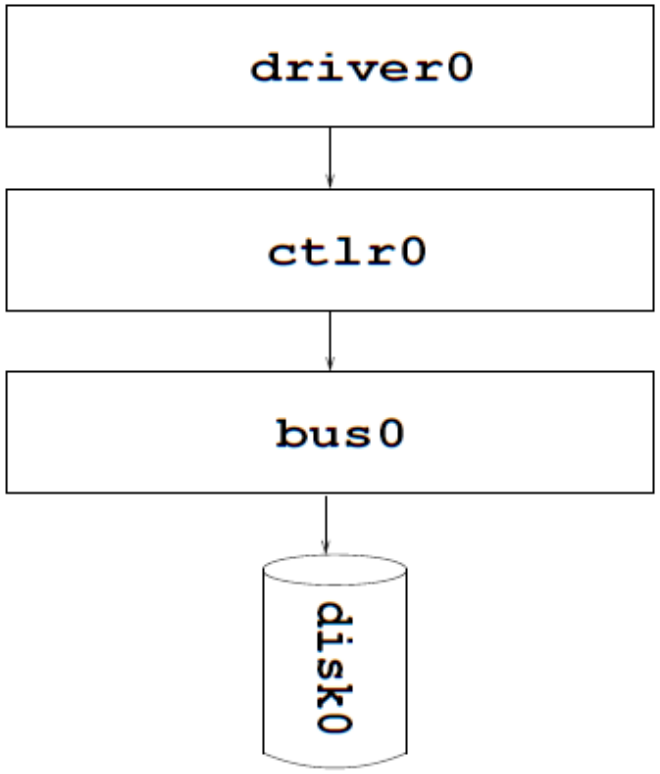

Fig. 3.1 **Topologia Sistemului simulat**

3.2 Workload

Workload-ul sau volumul de lucru, reprezintă sarcina totală pe care o vom lansa asupra sistemului cu scopul de a analiza comportamentul lui în timp.

În fişierul de workload avem în principiu 5 parametri ce descriu în mare parte o cerere către disc. Aceştia sunt:

- **Timpul de sosire a cererii:** Această valoare este un număr real şi specifică timpul după care cererea işi va face apariţia de la începutul simulării. Începutul simulării se consideră la momentul de timp 0.0. O altă specificaţie este faptul că aceste cereri trebuie să se afle întro ordine cronologică. Această valoare este exprimată în milisecunde.
- **Numărul discului**: Număr întreg ce specifică exact către ce dispozitiv va fi lansată cererea. Util atunci când avem o multitudine de discuri la dispoziţie şi dorim simularea unui sistem RAID. Vom folosi doar un disc pentru simulări deci acest număr va fi 0 tot timpul, însemnand discul 0.
- **Numărul blocului**: Număr întreg ce va indica adresa cererii pe disc la nivelul fizic. Această valoare nu este tot timpul aceasi, ci diferă în funcţie de maparea folosită de către simulator. În cazul nostru am folosit LBA.
- **Dimensiunea cererii**: Valoare întreagă ce ne indică dimensiunea unei cereri după numărul de blocuri ocupate. Din nou, este o valoare ce diferă în funcţie de metoda de adresare al simulatorului.
- **Fanioanele cererii**: Valoare în hexazecimal întreagă, ce ne va indica tipul de cerere. De exemplu, pentru bitul 0, vom avea o cerere de tip citire iar pentru valoarea 1, vom avea o cerere de tip scriere.

În continuare vom prezenta un model de workload personalizat cu scopul de a ne pune în evidenţă eficienta fiecărui algoritm în parte. Au fost generate cereri concurente distribute după o gaussiană cu medii variabile dar cu distribuţie mică.

```
0.000000        0       2458317         4       0
0.004000        0       16683180        4       0
80.789000       0       3962268         12      1
151.685539      0       2458317         16      1
152.469539      0       16683180        4       0
230.017539      0       2924420         12      1
269.021680      0       2924276         4       1
305.225945      0       2458317         4       1
306.447739      0       16683180        4       0
422.725739      0       8692968         8       1
472.275304      0       8693084         8       1
541.354304      0       2458317         8       1
542.159304      0       16683180        8       1
617.854304      0       8693024         4       1
656.127304      0       8693028         4       1
```

***Fig 3.2* Model de Workload**

După cum am specificat, la momentul începerii simulării, avem prima cerere lansată către disc la timpul 0.0 . Prima cerere este adresată discului 0 la adresa 2458317, cu o dimensiune de 4 blocuri şi este o cerere de citire.

După nicio milisecundă va interveni cea de a doua cerere la adresa 16683180. Deoarece timpul de apariţie între cele două cereri este atât de mic, putem considera cererile ca fiind concurente la disc.

Scopul acestui workload, este de a genera cereri concurente pe piste diferite ale discului.

Deasemenea, au fost create mai multe workload-uri cu distanţa dintre pistele accesate din ce în ce mai mare cu scopul de a analiza şi compara comportamentul algoritmilor.

O altă caracteristică al workload-ului este faptul că, cererile sunt condiţionate de timp, din nou vom putea analiza numărul de cereri ce vor fi servite în timp util, înaintea expirării termenelor limită.

3.3 Fişierul Output

La începerea unei simulări, majoritatea parametrilor sunt copiaţi la începutul fişierului output. Restul informaţiilor în acest fişier sunt referitoare la statisticile simularii, incluzând atât caracterisiticile a workload-ului folosit (dacă acesta a fost generat automat) cât şi indicatori la componentele discului simulat. Fiecare linie din acest fişier este unică iar cautarea parametrului dorit este cea mai uşoară metodă de a gasi rezultatele.

Disksim colecţionează un număr mare de statistici despre componentele de stocare a sistemului. Un alt beneficiu al simulatorului este faptul că putem elimina parametrii ce nu sunt necesari din fişierul output. În fishierul principal cu parametrii sistemului, putem bifa ce parametri dorim sa citim.

În continuare vom prezenta caţiva parametri legati de unitatea centrală de procesare, pentru a determina performanţa sistemului:

CPU Total idle miliseconds: suma timpilor de inactivitate pentru toate procesoarele simulate

CPU Time-Critical request count: numarul total de cereri generate ce sunt criticale în timp.

CPU Time-Limited request count: numărul total de cereri generate, ce vor fi limitate de timp.

Statisticele discului sunt scrise în fişierul output în acelaşi mod ca şi parametrii legaţi de unitatea centrala de prelucrare. Aceştia sunt :

Disk Seeks of zero distance : numărul de cereri ce nu au avut nevoie de timp de căutare. Cu alte cuvinte, următoarea cerere se afla pe aceaşi pistă.

Disk Seek distance stats: statistica pentru distanţa de căutare măsurată pentru cererile la disc.

Disk Full rotation time: timpul necesar unei rotaţii complete a discului.

Disk Transfer time stats: statistică pentru timpul de transfer necesar cererilor de a accesa discul.

Disk Positioning time stats: timpul necesar braţului de citire/scriere pentru poziţionarea sa.

Timpul de acces la date este un factor important pentru discurile HDD. Majoritatea timpilor de această natura sunt datorată caracteristicilor mecanice a discurilor rotative şi mişcării braţelor de citire/scriere. Timpul de căutare reprezintă o măsuratoare din momentul începerii miscării braţului de citire/scriere până când acesta se poziţionează pe pista potrivită unde se află cererea. În acest timp este inclus şi latenţa de rotaţie deoarece, sectorul ce trebuie accesat trebuie să se afle sub braţ. Aceşti doi timpi sunt exprimaţi în milisecunde.

Rata de transfer a biţilor, odată ce braţul se află în poziţia corectă, crează o întârziere în funcţie de numărul de blocuri transferate (deobicei de câteva blocuri). În cazul unui transfer de mai multe blocuri, cum sunt unele fişiere mai voluminoase, timpul de transfer poate creşte. Timp de întârziere poate apărea şi atunci când discurile sunt oprite din motivul de a economisi energie.

Timpul mediu de acces, al unui HDD, este timpul mediu de access care, în mod tehnic vorbind, numărul total de căutări posibile supra numărul total de căutări. Totuşi, în practică, acest timp mediu de căutare este determinat prin metode statistice sau prin simpla aproximare a timpului de căutare pe pistele discului.

Defragmentarea este o procedură folosită cu scopul de a minimiza întârzierile atunci cand avem cereri de acces la disc. Idea de bază, este mutarea datelor pe zone fizice cât mai apropiate. Unele sisteme de operare pot rula acest proces în mod automat, fiind invizibil utilizatorului. Totuşi în timpul defragmentării, sistemul va prezenta performanţe reduse, aşadar, trebuie găsit un moment optim când trebuie să lansăm o procedură de fragmentare cu scopul de a nu deranja un utilizator sau alte procese ce consumă multe resurse.

Latenţa este timpul pierdut pentru o rotaţie a discului când acesta trebuie să aduca sectorul sub braţul de citire/scriere. În funcţie de viteza de rotaţie putem determina o latenţă medie a discului. Viteza de rotaţie se măsoara în revoluţii pe minut.

Tabel 3.1 Viteza de rotaţie şi latenţa în HDD-uri

Viteza de rotaţie [rpm]	**Latenţa medie [ms]**
15,000	2
10,000	3
7,200	4.16
5,400	5.55
4,800	6.25

Putem calcula viteza de rotaţie unui disk dacă ştim latenţa medie.

Presupunem că latenţa medie este de 3ms. Aşadar o rotaţie completă va fi facută în 6ms. Întradevăr această valoare este dublă deoarece se face o medie între timpi. Limitele intervalului este 0ms, atunci când cererea se află fix sub braţul de citire/scriere şi 6ms atunci când cererea se află cu un bloc în spatele braţului.

Rata de transfer al datelor este o altă caracteristică importantă al unui HDD. Majoritatea discurilor cu o viteză de 7200 RPM au o rată de transfer de 1.030 Mbits/sec susţinută de 'disk-to-buffer'. Această rată depinde în principal, de poziţia pistei, rata fiind mai mare spre exteriorul discului decât pe interiorul său. Rata de transfer este mai mare pe zonele de exterior deoarece avem mai multe sectoare pe o pistă exterioară.

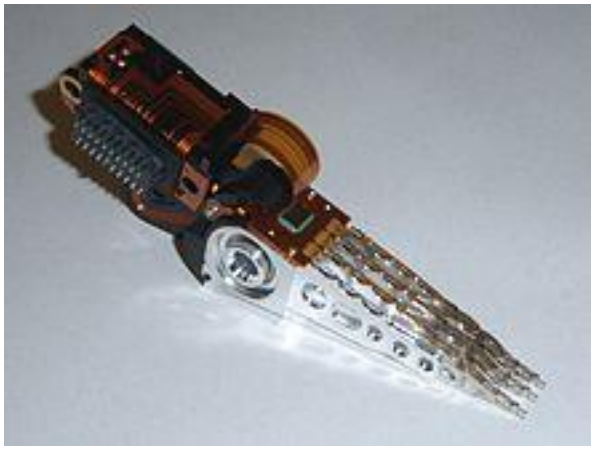

Fig 3.3. Braţ de citire/scriere

Transferul de date pe un HDD depinde, cum am menţionat mai sus, de viteza rotaţionala a discului şi de densitatea de date ce a fost inscripţionată. Datorită căldurii intense şi vibraţiilor puternice, ce produc limitări în performanţele discului, oamenii de ştiintă încearcă să dezvolte noi metode de imbunătăţire al transferului de date. Viteze mai mari înseamnă motoare mai puternice care la rândul lor vor genera căldură în exces. O altă metodă de îmbunătăţire este de a mări numărul de blocuri aflate pe un disc dar în acelaşi timp mări şi viteza de rotaţie. Trebuie găsit un compromis pentru o performanţă optimă.

Sursa: http://en.wikipedia.org/wiki/Hard_disk_drive

4. Rezultatele experimentale

În simulările de mai jos au fost notate, în principal, distanţa parcursă de către braţul de citire/scriere la nivel de blocuri.

Au fost extrase din fişierele output, din fiecare simulare, doi parametri principali pentru a putea determina performanţa sistemului.

În tabelul de mai jos putem observa pe fiecare coloană numărul de blocuri parcuse, numărul de cereri servite în timp util şi timpul mediu de acces. Discul având un numar total de blocuri de aproximativ 17.000.000, am hotărât să îndepărtăm pistele între ele, apoi sa măsuram performantele.

De reţinut este faptul că sistemul a primit 200 de cereri în total, dintre care aproximativ 100 din ele au fost concurente, distribuite dupa cum se poate observa în tabel. Restul cererilor au fost generate în mod aleator dealungul pistelor cu scopul de a ne apropia cât mai mult de realitate.

Tabel 4.1 Rezultatele algoritmului FCFS

Distanţa [10^6 blocuri]	0.6	1.6	5.6	9.6	11.6	13.6	15.6
Nr. de cereri servite	156	156	154	151	150	150	148
Timpul mediu de căutare[ms]	5.2637	5.25568	5.21261	5.36371	5.44408	5.59927	5.64785

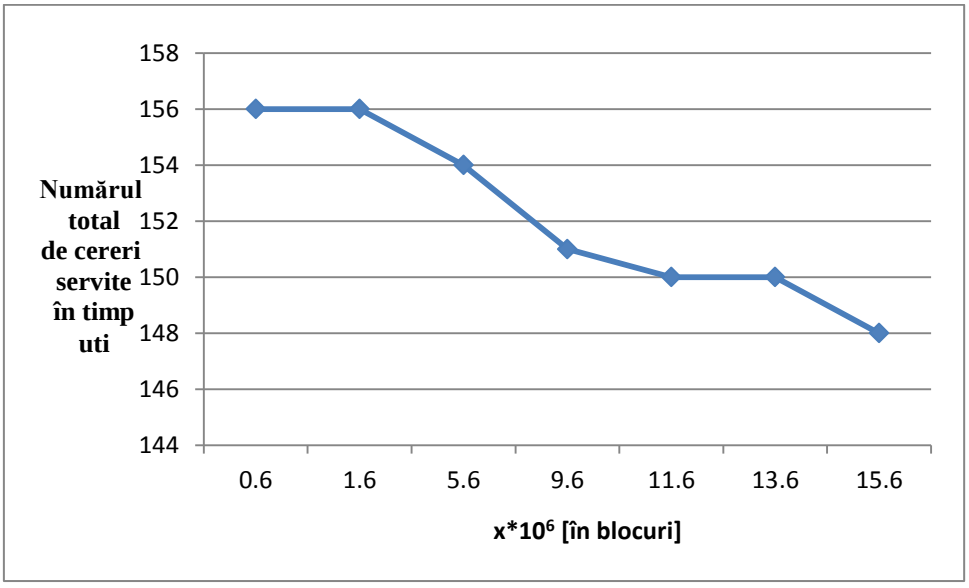

Fig. 4.1 **Numărul de cereri servite în cazul algoritmului FCFS**

După cum se poate observa din *Fig 4.1*, algoritmul FCFS prezintă performanţe destul de slabe. Acest lucru se datorată faptului că ordinea aleasă este cea cronologică a cererilor. La distanţă mică, avem un numar mai mare de cereri ce au fost servite în timp util. Cu cât creştem distanţa între piste putem observa cum tot mai multe cereri sunt pierdute datorită dependenţelor de timp.

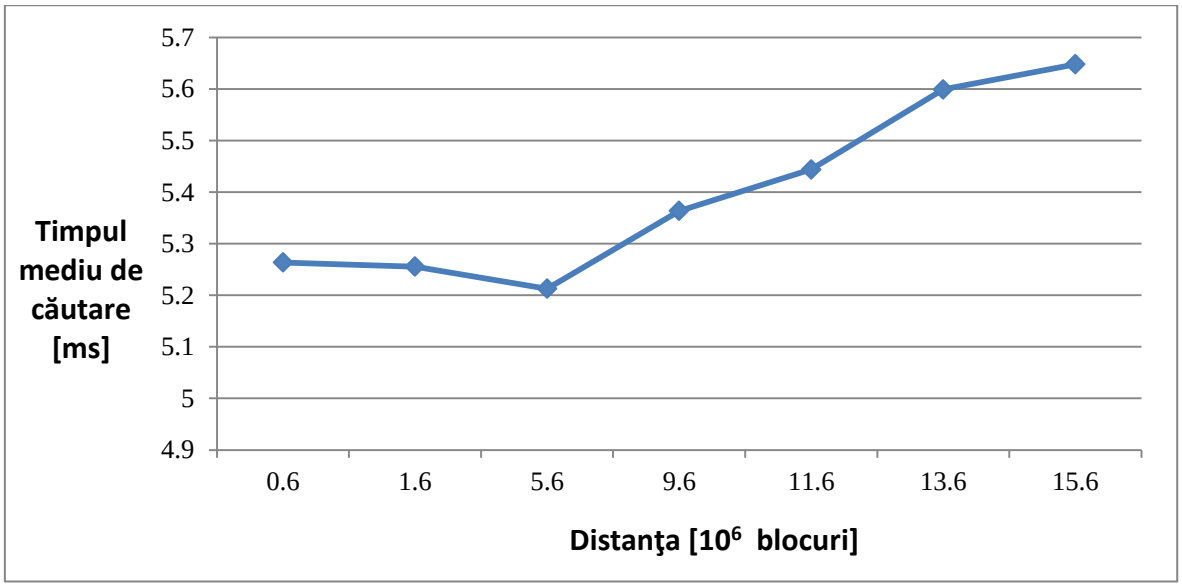

***Fig. 4.2* Timpul mediu de căutare al algoritmului FCFS**

În Fig. 3.2, putem observa timpul mediu de căutare al algoritmului. După cum ne aşteptam, timpul de căutare va creşte odată cu distanţa parcursă de către braţul de citire/scriere.

În continuare prezentăm algoritmul SSTF. Workload-ul a fost acelaşi, ca şi numărul de cereri ce au fost trimise la disc. Acest algoritm va alege cererile din coadă în funcţie de timpul de căutare cel mai scurt.

***Tabel 4.2.* Rezultatele algoritmului SSTF**

Distanţa [10^6 blocuri]	0.6	1.6	5.6	9.6	11.6	13.6	15.6
Nr. de cereri servite	189	187	188	190	189	187	187
Timpul mediu de căutare [ms]	3.02136	2.88343	3.11763	3.00791	2.93102	2.60026	2.67991

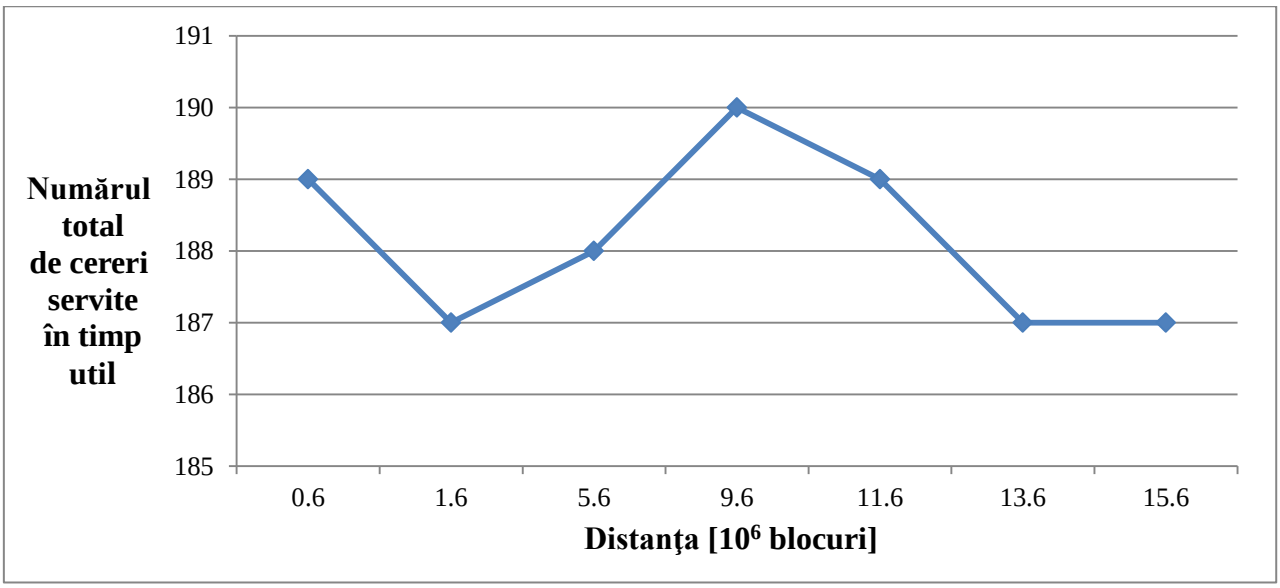

Fig. 4.3 **Numărul de cereri servite în cazul algoritmului SSTF**

Din start putem observa o îmbunătaţire asupra performanţelor folosind acest algoritm. Un numar destul de mare de cereri au fost servite în timp util chiar şi atunci când braţul a fost nevoit să parcurgă o distanţă considerabilă.

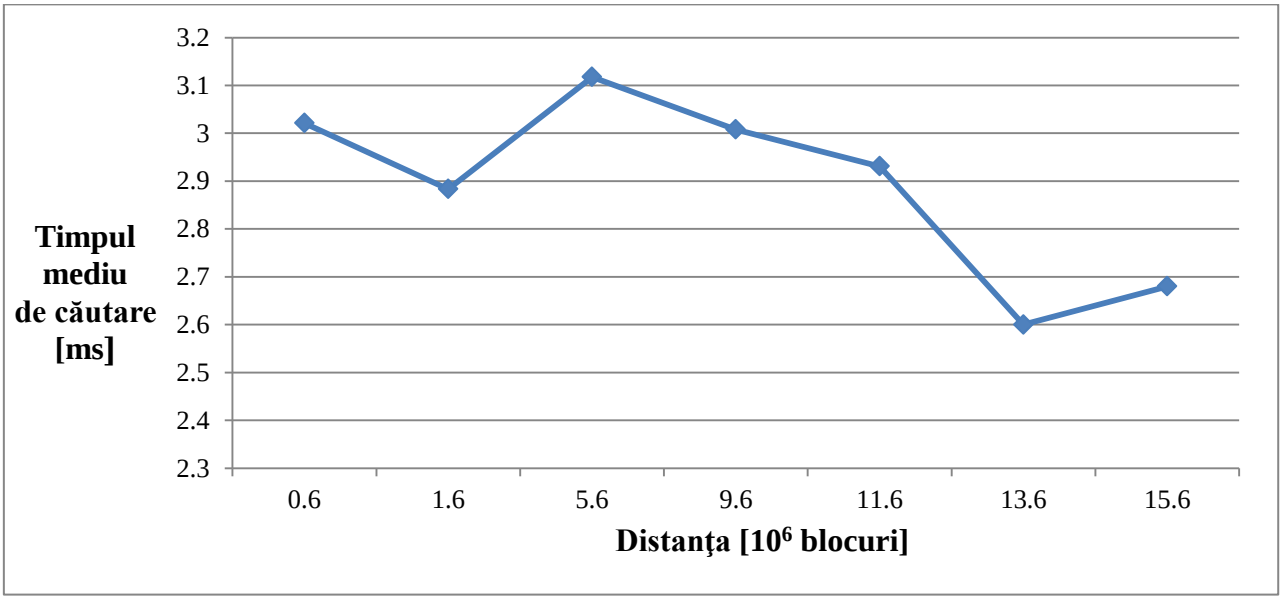

Fig. 4.4 Timpul mediu de căutare al algoritmului SSTF

Din Fig. 3.4 observa timpul mediu de căutare. Putem compara cu rezultatele anterioare şi să observăm o diferenţa majora. Timpul de acces este mult mai mic deoarece, algoritmul a ales cererile din coadă în funcţie de cel mai scurt timp de acces.

Algoritmul utilizat în acest scenariu, a fost SCAN. Din nou, observăm o îmbunătaţire a rezultatelor. Numărul total de cereri servite a crescut, iar în cel mai bun caz s-au pierdut 4 cereri datorită nerespectării termenelor limită de timp.

Tabel 4.3. **Rezultatele algoritmului Elevator (SCAN)**

Distanţa [10^6 blocuri]	0.6	1.6	5.6	9.6	11.6	13.6	15.6
Nr. de cereri servite	195	191	192	192	194	196	195
Timpul mediu de căutare [ms]	3.34077	3.0524	3.25205	3.13269	3.04355	3.28275	3.15631

Timpul mediu a de căutare a crescut dar în acelaşi timp mai multe cereri au fost servite la timp. Acest lucru poate fi confirmat deoarece, algoritmul SCAN a reuşit să servească cererile mai îndepartate de braţul de citire/scriere spre deosebire de algoritmul SSTF.

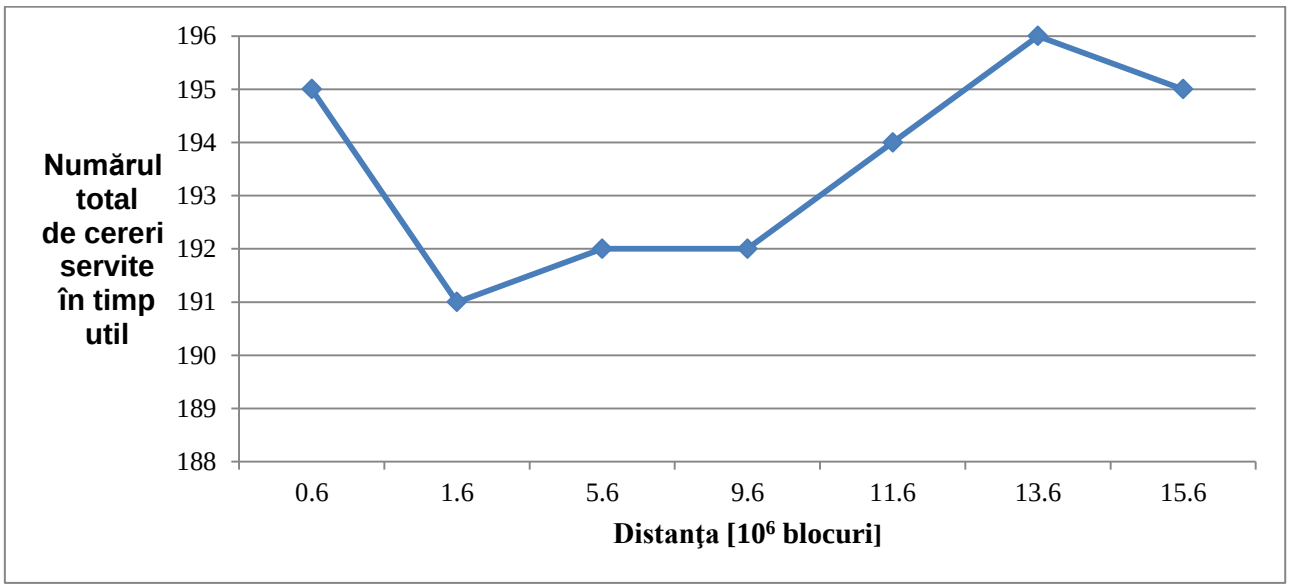

Fig 4.5. **Numărul de cereri servite în cazul algoritmului SCAN**

Algoritmul SCAN prezintă o performanţă destul de bună atunci când avem distanţe mari între cererile concurente şi atunci când cererile sunt foarte apropiate. Performanţa este ridicată în cazul în care cererile sunt alăturate deoarece braţul de citire/scriere nu trebuie să se deplaseze mult, cererea aflandu-se în direcţia de miscare a braţului. Este posibil ca o multitudine de cereri generate să fi ajuns la momentul în care braţul se afla pe direcţia potrivită servindu-le pe loc.

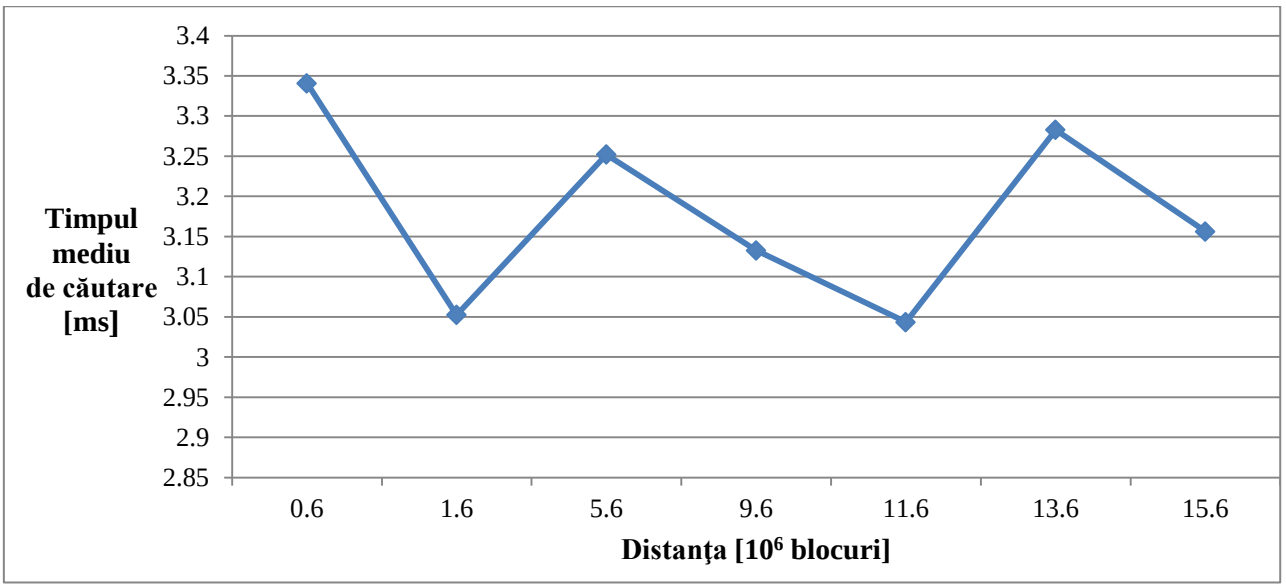

***Fig 4.6.* Timpul mediu de căutare al algoritmului SCAN**

Următorul algoritm prezentat va fi SDF. Principiul de sortare al acestui algoritm este căutarea cererii cererii următoare care este cea mai apropiată de braţul de citire/scriere. Distanţa de care vorbim este una fizică, aşadar vorbim de mişcarea mecanică a braţului. Nu trebuie făcută confuzia între timpul de căutare, Seek Time şi distanţa fizică.

Următorul tabel ne va prezenta rezultatele simulărilor:

***Tabel 4.4.* Rezultatele algoritmului SDF**

Distanţa [10^6 blocuri]	0.6	1.6	5.6	9.6	11.6	13.6	15.6
Nr. de cereri servite	196	196	190	198	193	194	190
Timpul mediu de căutare [ms]	3.19718	3.19321	3.14579	3.2327	2.94149	2.92568	3.13566

Observăm cazul cel mai favorabil, în care au fost servite în timp util 198 de cereri. Un alt caz favorabil se poate observa la distanţe mici între cererile generate. Deoarece SDF-ul va ordona coada de servite după cea mai mică distanţă, este de aşteptat ca cererile apropiate să fie servite întrun timp mai scurt fără a se pierde timp pe căutare.

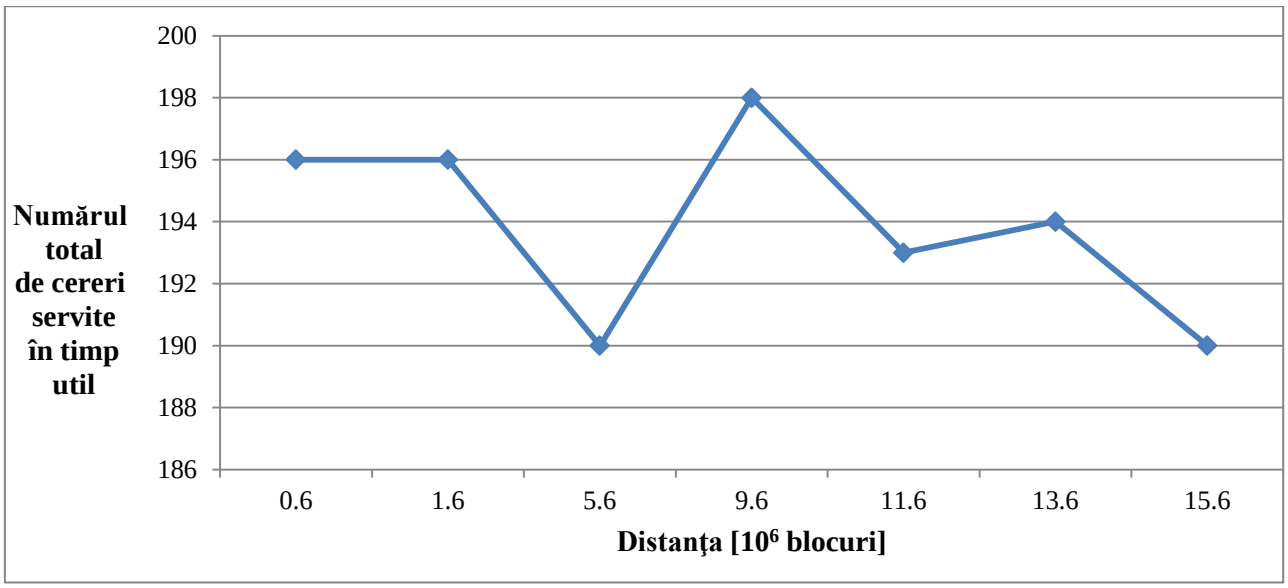

Fig. 4.7. **Numărul de cereri servite în cazul algoritmului SDF**

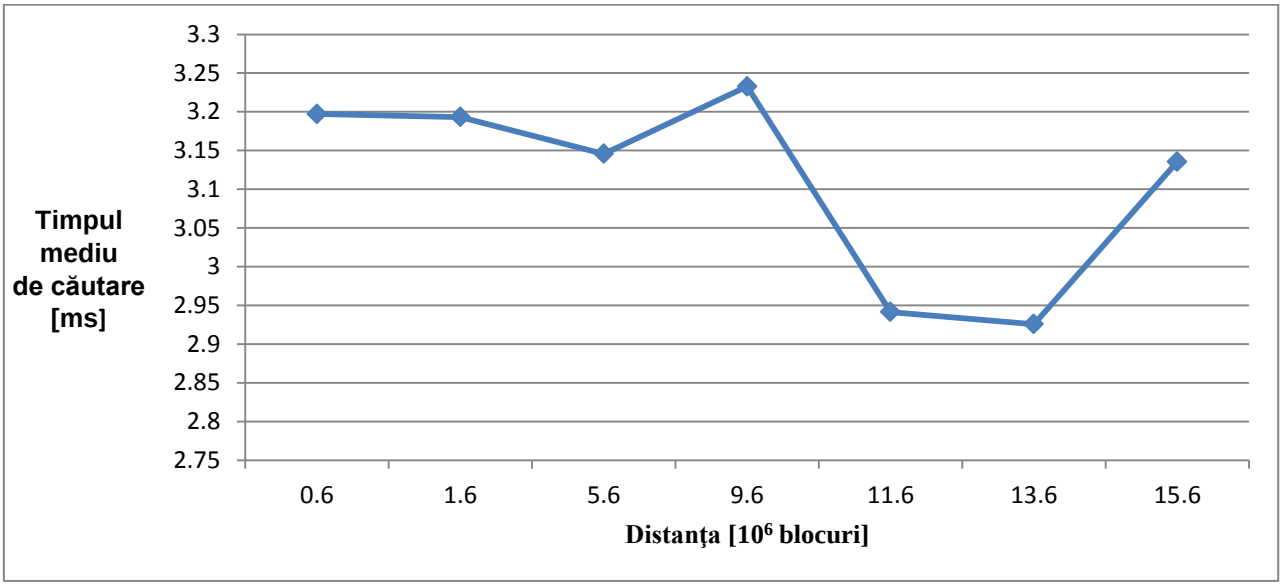

Fig. 4.8 Timpul mediu de căutare al algoritmului SDF

Cel de al doilea caz în care avem rezultate bune, este acela în care cererile generate sunt pe pista din mijlocul discului. Acest lucru se explica uşor deoarece odată ce braţul de citire/scriere se află la jumătatea discului, în orice direcţie de apariţie a cererii, timpul va fi acelaşi.

În concluzie, aces algoritm este util şi prezintă o performanţă remarcabilă atunci când cererile sunt generate în jurul pistei din mijlocul discului.

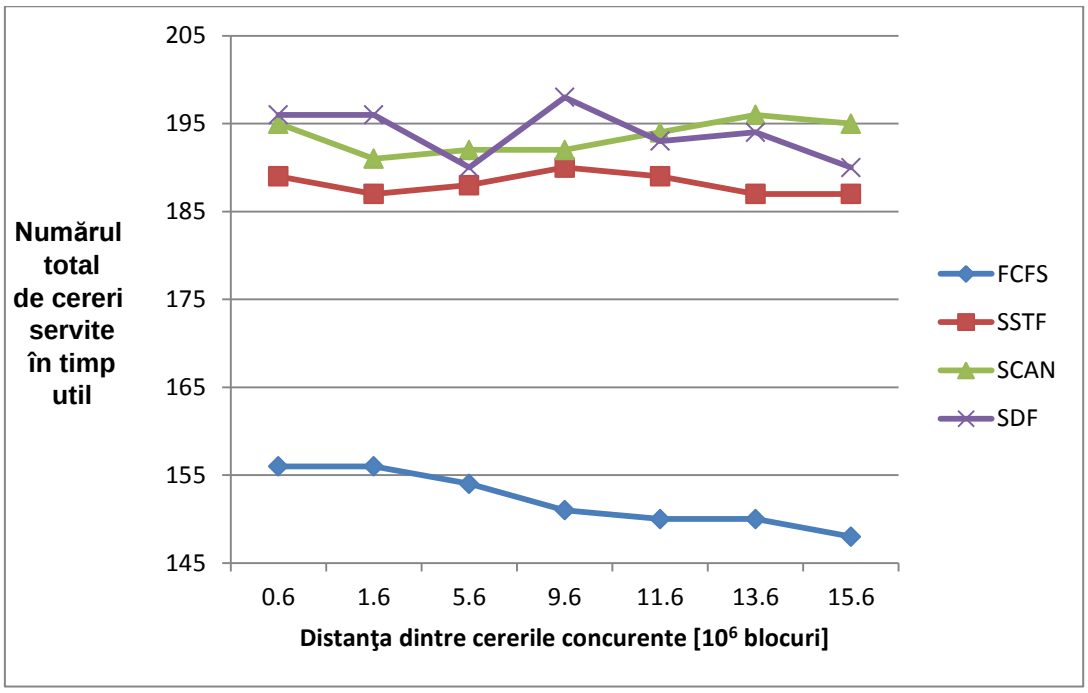

Fig. 4.9. **Numărul de cereri servite în cazul celor 4 algoritmi simulaţi**

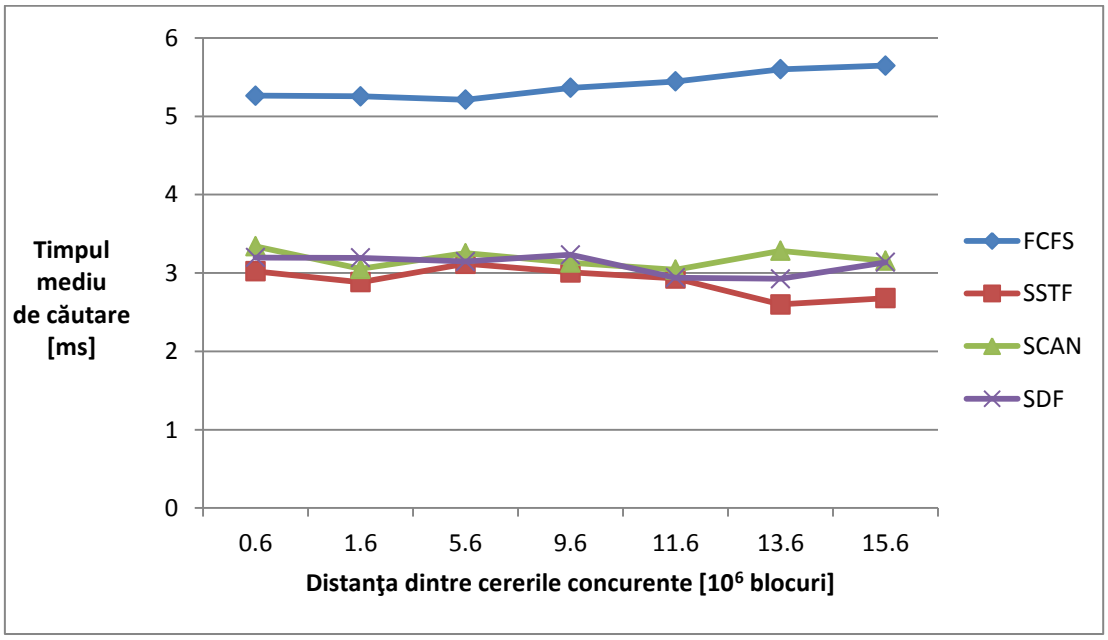

Fig. 4.10. Timpul mediu de căutare a celor patru algoritmi

Concluzii

Obiectivul proiectului de diplomă a fost determinarea unui criteriu şi un algoritm optim corespunzător de acces la disc în timp real, pentru cazurile de concurenţă în accesul de date de pe disc din partea a două sau mai multe procese cu conditionări de limită de timp.

Initial, au fost explicate metodele de funcţionare ai acestor algoritmi în primele capitole.

În următoarea etapă au fost simulaţi aceşti algoritmii după mai multe scenarii cu scopul de a determina criteriile de performanţă. S-a putut observa o îmbunătăţire a timpului de căutare atunci când a fost folosit algoritmul SSTF dar cu riscul de a pierde cereri, aşadar un compromis între cele două masurători trebuie găsit.

Un alt algoritm testat cu performanţe remarcabile a fost SDF. În urma simulărilor am tras concluzia că acest algoritm reuşeşte o performanţă bună dacă tot mai multe cereri sunt adresate pistelor de la jumătatea discului. Deoarece braţul lucrează mai mult pe mijlocul discului, acesta are o distanţă egala spre capetele discului reuşind să reducă timpul de căutare şi distanţa parcursă.

Ca o continuare a proiectului, putem implementa un software, gestionat de sistemul de operare, cu scopul de a modifica politica de planificare în funcţie de statisticele cererilor ce ajung la disc. Sistemul de operare va putea să schimbe această politică după adresele şi tipul cererilor. O fereastră de analiză poate fi creată cu primele n cereri, apoi putem crea statistica bazată pe fereastra respectivă.

Bibliografie

[1] http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html

[2] Arezou Mohammadi and Selim G. Akl, Scheduling Algorithms for Real-Time Systems, Technical Report No. 2005-499, July 15, 2005

[3] Kartik Gopalan, Real-Time Disk Scheduling Using Deadline Sensitive SCAN, 2001

[4] Shenze Chen, John A. Stankovic, James F. Kurose, Don Towsley, Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems, 1991

[5] http://en.wikipedia.org/wiki/Concurrency_control

[6] Carl Staelin, Gidi Amir, David Ben-Ovadia, Ram Dagan, Michael Melamed, Dave Staas, Real-time disk scheduling algorithm allowing concurrent I/O requests, 2009

[7] John S. Bucy, Jiri Schindler, Steven W. Schlosser, Gregory R. Ganger, he DiskSim Simulation Environment Version 4.0 Reference Manual, May 2008

[8] http://en.wikipedia.org/wiki/Defragmentation

[9] Intel Corporation and Seagate Technology, Serial ATA Native Command Queuing, July 2003

[10] http://faculty.plattsburgh.edu/jan.plaza/teaching/papers/seektime.htm