Librăria online *ReadsProfiler*

Sebastian Ciobanu

Facultatea de Informatică,
Universitatea *Alexandru Ioan Cuza*, Strada Gen. Berthelot 16, Iaşi
sebastian.ciobanu@info.uaic.ro
http://www.info.uaic.ro/

Abstract. Combinând elemente de reţele de calculatoare (TCP implementat concurent prin crearea cate unui fir de execuţie pentru servirea fiecărui client şi prin folosirea socketţilor BSD), de baze de date (MySQL ca modalitate de stocare a datelor despre cărţile disponibile într-o librărie) şi de algortimică/statistică (algoritmi de recomandare de tip filtrare colaborativă bazată pe utilizator sau filtrare bazată pe conţinut, similaritatea cosinus), aplicaţia *ReadsProfiler* de tip client/server se ocupă de accesul la o librărie online, oferind recomandări de cărţi utilizatorilor.

Keywords: TCP, MySQL, cosine similarity, filtering, e-bookshop

1 Introducere

În ziua de astăzi, aproape orice magazin online are o modalitate sau mai multe de a îşi convinge proprii clienţi să cumpere noi produse. Una din aceste metode este reprezentată de sistemul de recomandări.

Scopul unui astfel de sistem este de a genera recomandări semnificative unei colecţii de utilizatori pentru obiecte sau produse care ar putea să îi intereseze. Să considerăm doar un exemplu minimal al unei librării online: utilizatorul Negrescu Vasile a căutat cărţile: *Ion*, de Liviu Rebreanu; *Răscoala*, de Liviu Rebreanu; *Pădurea spânzuraţilor*, de Liviu Rebreanu. Se poate intui că lui Vasile îi plac cărţile lui Liviu Rebreanu, aşa că sistemul poate *urma* această regulă şi îi va putea recomanda cartea: *Ciuleandra*, de acelaşi autor, carte de care Vasile, probabil, este interesat. Exemple din lumea reală ar include: sugestiile pentru cărţi de pe Amazon, sugestiile de filme pe netflix etc.

Aplicatia dezvoltată este axată pe 2 termeni cheie: librărie online şi sistem de recomandări. Aceasta se numeşte *ReadsProfiler* şi este o aplicaţie client-server prin care se permite accesul la o librărie online. Oferind o experienţă tradiţională prin **clasica interfaţă** cu utilizatorul, aplicaţia este **simplu de utilizat**, conţine doar **cărţi scrise sau traduse în limba română**, iar *tehnicile* utilizate pentru sistemul de recomandare sunt ideale pentru date relativ mici (faţă de numărul de produse de pe Amazon, spre exemplu) aşa că **implementarea** poate fi de folos unui novice în crearea propriului sistem de recomandări. Un alt avantaj este faptul că programul este **gratis**, open-source.

2 Tehnologiile utilizate

În cadrul aplicaţiilor în reţea **paradigmele** folosite cel mai frecvent sunt: modelul client/server, RPC şi P2P. În cadrul primei paradigme, un server oferă servicii clienţilor. Serverul poate satisface cererile provenite de la clienţi în mod **iterativ sau concurent**. De obicei, serverele concurente folosesc protocoale de comunicaţie orientate conexiune. În acest caz, cererile sunt procesate concurent.

În *ReadsProfiler*, s-au folosit:

- paradigma: modelul client/server

- server: concurent

- protocol (la nivelul transport): TCP

Serverul a fost ales să fie **concurent** din considerente practice: când se doreşte a cumpăra un produs şi clientul este pus să aştepte, rezultatul este dat de clientul care renunţă la ideea de a mai cumpăra ceva.

TCP a fost ales pentru că este un protocol de transport orientat conexiune, fără pierdere de informaţii: dacă un client caută o carte şi primeşte un rezultat neavând legătura cu acea căutare, acesta îşi pierde încrederea în magazinul online şi astfel magazinul va avea de suferit; mai grav este momentul în care se descarcă o carte (pentru care poate s-a şi plătit) şi unele părţi (sau mai multe) nu sunt ceea ce trebuie (citibile); deşi UDP oferă o mai mare viteză, preferăm să aducem rezultate corecte într-un timp mai *lung* decât cel dat de folosirea UDP-ului (care în practică de multe ori nici nu este observabil); în astfel de situaţii (magazin online), doar TCP-ul poate fi adus în discuţie. Ca **interfaţă de programare a aplicaţiei**, am ales **socketii BSD**.

Am ales ca datele despre cărţi (şi nu numai) să fie stocate într-o baza de date **MySQL**. Faţă de SQLite, MySQL oferă **acces în reţea la baza de date şi un grad mare de concurenţă** (paralelism) a interogărilor, lucruri chiar necesare unei aplicaţii în reţea precum este cea de faţă. Totodată, căutările frecvente ale utilizatorilor după diverse criterii (combinate) se pliază perfect în acest context al bazelor de date. Iarăşi, **volumul de date** este un alt motiv pentru care am ales această opţiune.

3 Arhitectura aplicaţiei

3.1 Adăugiri la interfaţa BSD

Pentru că în cadrul stivei de protocoale TCP/IP fiecare nivel oferă servicii nivelului superior, avem că nivelul transport (unde am folosit protocolul TCP) oferă servicii nivelului aplicaţie. Astfel, ca programatori, la nivelul aplicaţiei vom şti că biţii vor ajunge în cele din urmă la destinatar în ordine şi în mod eficient (să nu mai vorbim de celelalte *încercări* pe care le-am fi avut dacă nivelurile inferioare nivelului transport n-ar fi existat).

Totuşi, la nivel de aplicaţie, prin utilizarea interfeţei socketilor BSD, nu putem fi siguri că un apel *read* sau *write* va duce sigur la citirea/scrierea numărului de bytes specificat. Iarăşi, o altă problemă este *Dacă am format o conexiune între*

client şi server şi unul dintre cei doi o părăseşte nepoliticos, cum poate cealaltă parte să-şi dea seama că nu mai are cu cine comunica?. Evident, problema se pune doar dacă cele două părţi mai au ce comunica (*read, write*). Dacă nu, nu există nicio problemă. Problema se rezolvă uşor dacă partea rămasă se aşteaptă la un mesaj de la cealălaltă parte, pentru că apelul *read* îşi dă seama de acest lucru. Dacă însă partea rămasă transmite un mesaj, el va crede că mesajul a ajuns la destinatar cu succes (pentru că abia al doilea apel *write* îşi dă seama de *ruperea* conexiunii, nu şi primul).

Aceste două probleme au fost rezolvate la nivelul aplicaţiei prin implementarea funcţiilor *myRead*, *myWrite* şi *isPeerClosed*. Mai multe detalii vom avea în secţiunea următoare.

3.2 Subniveluri ale nivelului aplicaţie

Luând in calcul complexitatea implementării unei astfel de aplicaţii, am considerat că ar fi maximă nevoie de metode precum *mySendLength, mySendMessage, mySendFile, mySendPage, sendTable, myRecvLength, myRecvMessage, myRecvFile, myRecvPage, recvTable* care să faciliteze comunicarea între client şi server. Mai mult, în final *utilizatorul* funcţiei *main()* nu va avea acces decât la funcţii de tipul *login, find*, atât la nivelul clientului, cât şi la nivelul serverului (programatorul de la acest subnivel nu va fi nevoit să cunoască detalii de implementare ale acestor funcţii).

3.3 Proiectarea unui protocol de comunicaţie între client şi server cu scopul de a uşura comunicarea dintre cei doi

a) Definirea operaţiilor permise clientului (comanda, informaţii de furnizat, semantica)

La nivelul cel mai de sus (al clientului utilizator de terminal), am definit următoarele operaţii:

1)

1. comanda: **signup**
   informaţii de furnizat: *user, pass*
   semantica: va înregistra un nou utilizator cu numele *user* şi parola *pass*. Dacă *user* este înregistrat deja, se va afişa un mesaj corespunzător.

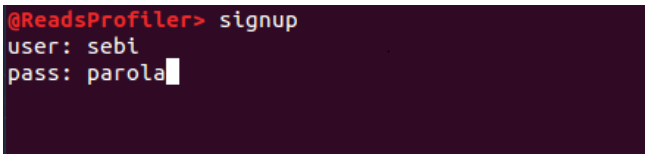

Fig. 1. Comanda **signup**

2. comanda: **login**
   informaţii de furnizat: *user, pass*
   semantica: va executa autentificarea utilizatorului. Dacă *user* nu este gasit în baza de date sau dacă parola nu corespunde, se va afişa un mesaj corespunzător. **Atentie!** Fără logare, majoritatea comenzilor sunt inaccesibile (doar **signup** e accesibilă).

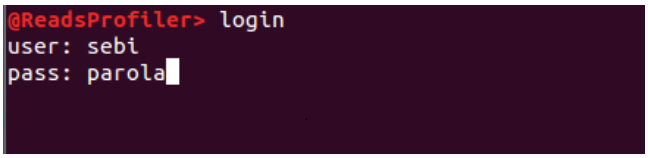

Fig. 2. Comanda **login**

3. comanda: **logout**
   informaţii de furnizat: -
   semantica: va executa delogarea utilizatorului curent. În cazul în care nu este nimeni logat, va afişa un mesaj corespunzător.

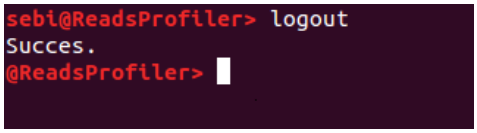

Fig. 3. Comanda **logout**

4. comanda: **quit**
   informaţii de furnizat: -
   semantica: va închide programul în 3 secunde.

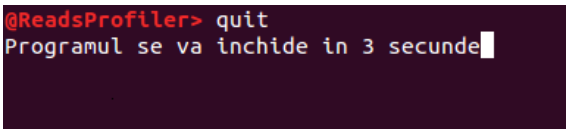

Fig. 4. Comanda **quit**

5. comanda: **find**
   informaţii de furnizat: criteriile de cătare, valorile criteriilor de căutare
   semantica: va duce la afişarea tuturor cărţilor ce îndeplinesc toate criteriile menţionate. Dacă nu există astfel de cărţi, se va afişa un mesaj corespunzător.

Ca ordine a operaţiilor, această comandă are sens doar dacă sunteţi logat şi nu aţi accesat chiar înainte vreo pagină a vreunei cărţi.

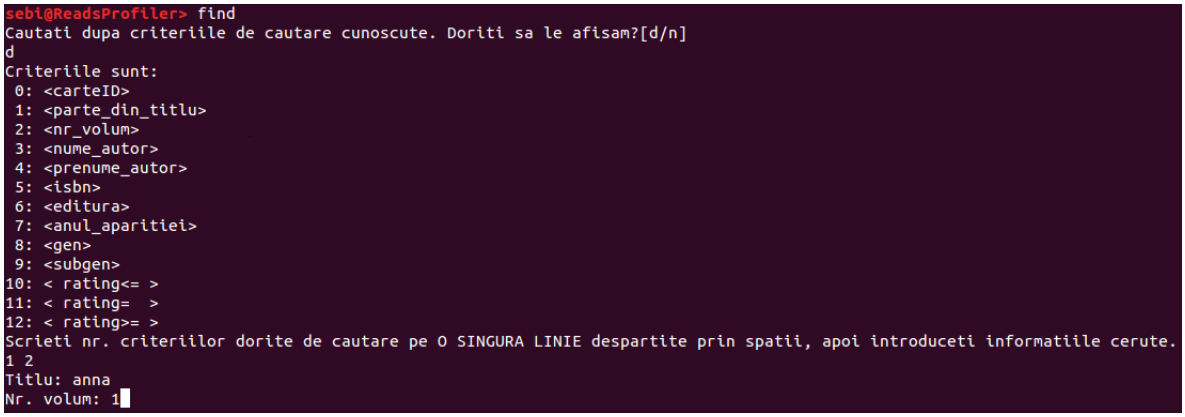

Fig. 5. Comanda **find**

6. comanda: **list**
   informaţii de furnizat: -
   semantica: va afişa maxim 100 de cărţile ce sunt disponibile în cadrul librăriei. Ca ordine a operaţiilor, această comandă are sens doar dacă sunteţi logat şi nu aţi accesat chiar înainte vreo pagină a vreunei cărţi.

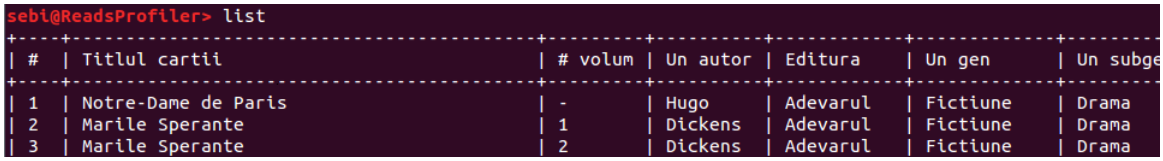

Fig. 6. Comanda **list**

7. comanda: **access**
   informaţii de furnizat: numărul cărţii din ultimul afişj (list/find) cu succes
   semantica: va accesa pagina aferentă cărţii cu numărul dat (nr. a apărut în ultimul apărut în ultimul afişaj list/find). Dacă nr. nu este potrivit, se va afişun mesaj corespunzător. Ca ordine a operaţiilor, această comandă are sens doar dacă sunteţi logat, nu aţi accesat chiar înainte vreo pagină a vreunei cărţi şi aţi folosit măcar o dată list/find.
8. comanda: **download**
   informaţii de furnizat: numele fişierului salvat

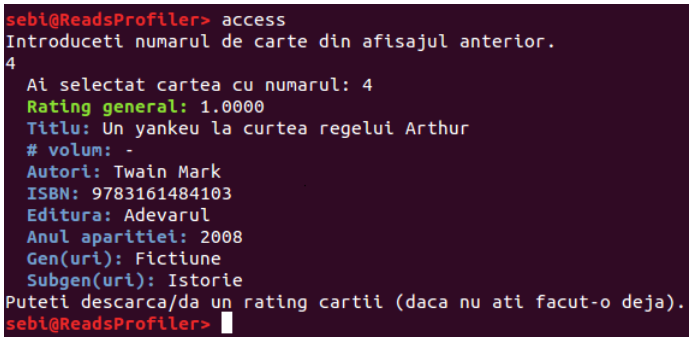

Fig. 7. Comanda **access**

semantica: va avea sens doar dacă ultima comandă folosită este **access**. Cartea accesată anterior va fi descarcată de utilizator la calea ../descarcari şi va avea denumirea menţionată.

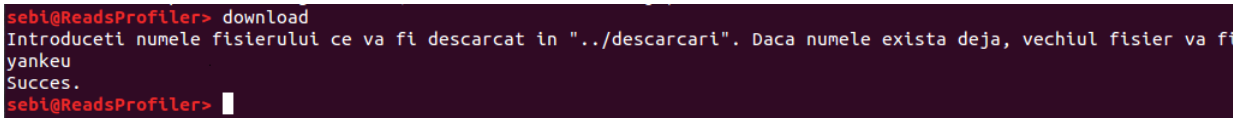

Fig. 8. Comanda **download**

9. comanda: **rate**
   informaţii de furnizat: ratingul dat carţii accesate ($0 \leq rating \leq 5$)
   semantica: va avea sens doar dacă ultima comandă folosită este **access**. Cărţii accesate anterior i se va atribuit ratingul menţionat doar dacă userul nu i-a mai dat vreun rating în trecut, caz în care se va afişa un mesaj corespunzător.

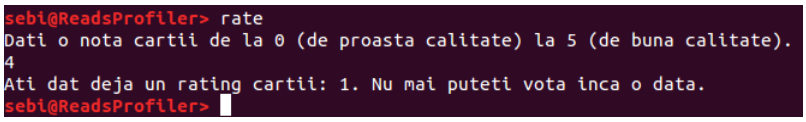

Fig. 9. Comanda **rate**

10. comanda: **back**
    informaţii de furnizat: -
    semantica: va avea sens doar dacă ultima comandă folosită este **access**. Ne va întoarce la rezultatul ultimei comenzi find/list.

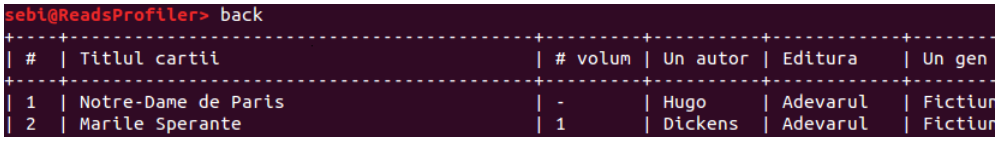

Fig. 10. Comanda **back**

11. comanda: **recommend**
    informaţii de furnizat: -
    semantica: va da 1 sau mai multe recomandări de cărţi userului curent. Ca ordine a operaţiilor, această comandă are sens doar dacă sunteţi logat, nu aţi accesat chiar înainte vreo pagină a vreunei cărţi şi aţi folosit măcar o dată list/find.

Fig. 11. Comanda **recommend**

12. comanda: **help**
    informaţii de furnizat: -
    semantica: va da informatţii despre comenzi şi reguli de aplicare a lor (inf. asemănătoare cu cele de aici).

b) Raportarea erorilor

Pe lângă erorile interne aferente fiecăărui participant la conexiune, mai apar erori *interesante* şi pentru partener. Eroarea cea mai distructivă dacă o putem denumi aşa este cea menţionată în subsecţiunea 3.1. Celelalte erori ce vor fi comunicate pe reţea vor fi:

- signup: User deja existent.
- login: Aţi greşit userul/parola.
- find: Nu există astfel de cărţi.
- rate: Aţi dat deja un rating cărţii. Nu mai puteţi vota încă o dată.

Celelate erori (cum ar fi neîncadrarea ratingului între 0 şi 5) vor fi tratate doar local, neimplicând şi partenerul. Pentru fiecare eroare vom avea perechea (cod de eroare, mesaj explicativ).

3.4 Stocarea informaţiilor

În cadrul bazei de date MySQL, este folosită o serie de tabele pentru stocarea datelor. Vom arunca doar o privire de ansamblu asupra lor:

- opere(opID,titlu,nr_volum)

- carti(isbn,editura,calea,an_aparitie,rating,opID)
- autori(autID,nume,prenume)
- opere_autori(opID,autID)
- ierarhie(subgen,gen)
- abordari(opID,subgen)
- history(carteID,user,accessed,downloaded,rating)
- recommendedTo(carteID,user)
- view-ul avgRating(carteID,avgrate)

3.5 Algoritmul de recomandare

Conform cerintzei, utilizatorilor li se oferă recomandări de cărţi. Vom folosi 2 metode de recomandare:

- **filtrare bazată pe conţinut**; astfel vom folosi faptul că avem cărţile sortate pe genuri şi că autorii abordează mai multe genuri în operele lor

- **filtrare colaborativă bazată pe utilizatori**; astfel, sistemul de recomandări se va putea îmbunătăţi în acurateţe, luând în considerare factori precum gusturile unui utilizator, preferinţele altor utilizatori cu gusturi similare, rating-ul pe care l-au primit cărţile de la clienţii cărora le-au fost recomandate.

Sisteme de recomandare bazate pe conţinut se recomandă articole al căror conţinut e similar cu cel al altor itemi agreaţi de user în trecut sau care se potriveşte cu atributele userului.

Sisteme cu filtrare colaborativă - unui utilizator i se recomandă itemi pe baza voturilor tuturor userilor din trecut.

Mai ales pentru filtrarea cea din urma va trebui să calculăm similaritatea dintre 2 utilizatori. Există numeroase idei de calcul (distanţa euclidiană, Manhattan etc.), dar pentru datele noastre, în care nu fiecare user a dat rating la aproape toate cărţile (cel mai probabil, niciunul nu a făcut aceasta), vom folosi similaritatea cosinus care se descurcă bine şi în caz de sistem bazat pe utilizator (cazul nostru). Pentru a înţelege mai bine, trebuie să ne imaginăm fiecare utilizator ca pe un vector n-dimensional, unde n=numărul de cărti din baza de date. Fiecare componentă a vectorului reprezintă ratingul dat de utilizator cărţii *de pe poziţia respectivă.*

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Fig. 12. Similaritatea cosinus [9]

Pentru a recomanda o carte lui X, calculăm similarităţile dintre X şi ceilalţi. Îl alegem pe cel ce are similaritatea dintre el şi X cea mai mare. Fie acesta Y.

Căutam o carte C căreia Y i-a dat rating *pozitiv* şi căreia X nu i-a dat rating. C va fi cartea de recomandat lui X. Procedeul anterior se va perfecţiona prin utilizarea primilor k (k fiind ştiut în prealabil) utilizatori similari cu X.

În cazul nostru, am folosit un algoritm de recomandare bazat pe conţi nut şi unul bazat pe useri. Dacă prin astfel nu găsim nicio recomandare luam **topul** cărţilor după rating ş recomandăm prima carte căreia userul curent nu i-a dat rate.

Algoritmul bazat pe conţinut sună astfel:

1. U=user curent
2. extragem cărţile din istoricul lui U cu ratingul ≥ 3
3. extragem autorii acelor cărţi
4. extragem subgenurile abordate de acei autori
5. extragem cărţile care abordează acele subgenuri şi cărora U nu le-a dat nici rate, nici download şi care au fost recomandate userilor
6. calculăm pentru fiecare carte media dată de notele acordate de userii cărora le-au fost recomandate
7. ordonăm cărţile după această medie
8. *returnăm* prima carte.

Algoritmul bazat pe useri sună astfel:

1. U=user curent=$(r_1, ..., r_n)$, n=nr. de cărţi, r_i=ratingul dat cărţii i de către U
2. formăm şi ceilalţi useri astfel
3. calculăm similaritarea dintre U şi fiecare dintre ceilalţi useri prin similaritatea cosinus
4. ordonăm userii după această similiaritate
5. alegem primul user = V
6. căutăm cărţi ale lui V neratingate de U
7. *returnăm* prima găsită.

4 Detalii de implementare

În cadrul protocolului de comunicatie client-server, la nivel de implementare, putem adauga:

- Formatul mesajelor:
  - de obicei, mesajele vor fi de tipul lungimea_mesajului + mesaj
  - în cazul diferenţierii erorilor de mesaje de succes (acolo unde trebuie trimise erori perechii de comunicare), lungimea va fi şi ea prefixată cu '0'-mesaj de succes, '1'-mesaj de eroare
  - un alt tip de mesaj este cel în care am trimis 0/1/2 recomandări clientului şi am prefixat astfel întreg mesajul cu '0'=0 rec., '1'=1 rec. 2=2 rec.
- la *download*: vom deschide fişierul în format binar ("rb","wb") şi apoi îl vom trimite; dacă îl deschideam în format normal ("r","w") era posibil să găsim un EOF pe la mijlocul fişierului şi transferul ar fi fost deci doar pe jumătate;

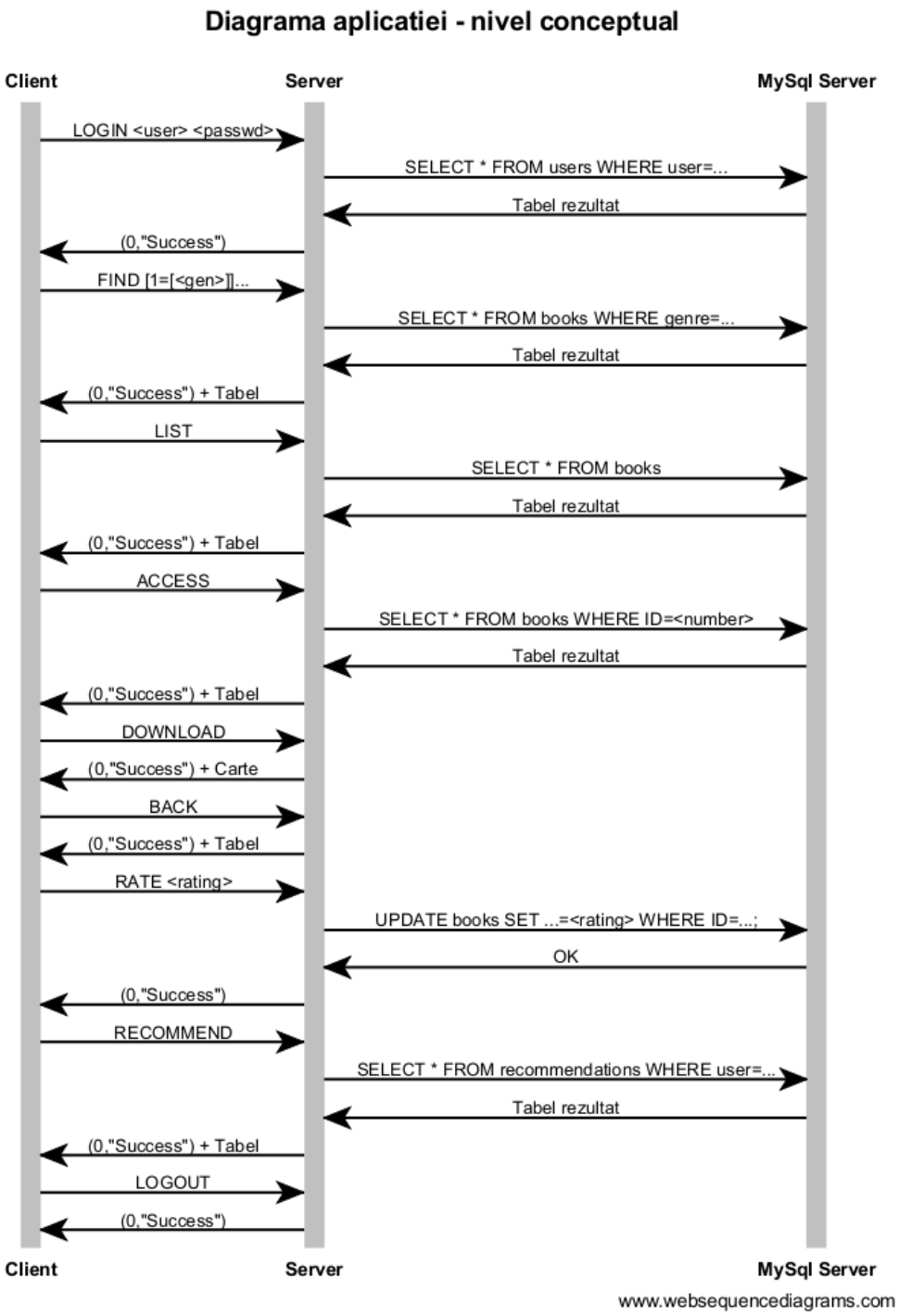

Fig. 13. Diagrama aplicaţiei - nivel conceptual

– trimiterea de mesaje face apel la următoarele primitive care asgiură citirea/scrirea numărului de bytes specificaţi:

Primitive citire/scriere implementate de utilizator:

```
ssize_t myWrite(int fd, const void *buf, size_t count)
{
    size_t left=count;
    ssize_t written=0;
    const char *charBuffer=buf;
    while(left>0)
    {
        if(isPeerClosed(fd)==1)
        {
            return -2;
        }
        written=write(fd,charBuffer,left);
        if(written==-1)
        {
            return -1;
        }
        left-=written;
        charBuffer+=written;
    }
    return count;
}

ssize_t myRead(int fd, void *buf, size_t count)
{
    size_t left=count;
    ssize_t beenRead=0;
    char *charBuffer=buf;
    while(left>0)
    {
        beenRead=read(fd,charBuffer,left);
        if(beenRead==-1)
        {
            return -1;
        }
        else if(beenRead==0)
        {
            return -2;
        }
        left-=beenRead;
        charBuffer+=beenRead;
    }
```

```
    return count;
}
```

– Interfaţa de programare: bazată pe socketBSD După cum am menţionat anterior, serverul va fi concurent. La nivel de implementare, vom folosi Pthread pentru UNIX.

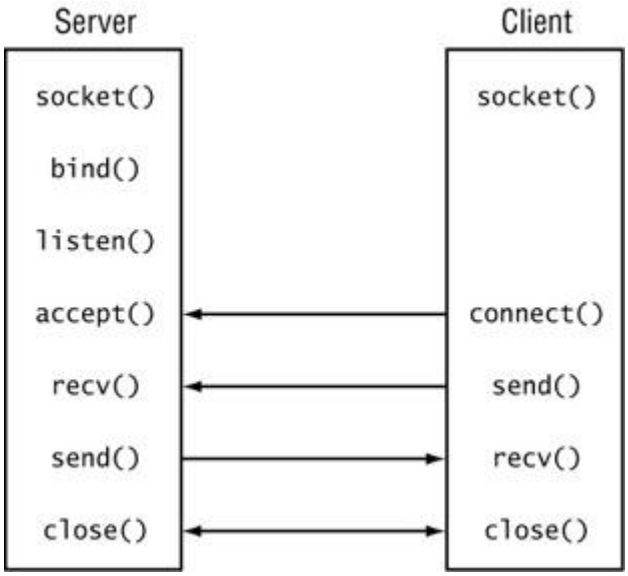

Fig. 14. Server concurent TCP [14]

login server vs login client:

```
void login(int fd,MYSQL *con)// server
{
    int length=myRecvLength(fd);
    char *user=malloc(length);
    myRecvMessage(fd,length,user);
    user[length]='\0';
    length=myRecvLength(fd);
    char *pass=malloc(length);
    myRecvMessage(fd,length,pass);
    pass[length]='\0';
    char aux[1000];
    sprintf(aux,"select * from login where user='%s' and pass='%s'",user,pass);
    free(user);
```

```
    free(pass);
    MYSQL_RES * res=myQuery(con,aux,fd);
    if(mysql_num_rows(res)==0)
    {
        mySendType(fd,'1');
        char *message="Ati gresit userul/parola.\n";
        mySendLength(fd,strlen(message));
        mySendMessage(fd,strlen(message),message);
    }
    else
    {
        mySendType(fd,'0');
        char *message="Succes.\n";
        mySendLength(fd,strlen(message));
        mySendMessage(fd,strlen(message),message);
    }
    mysql_free_result(res);
}

void login(int fd)// client
{
    if(flagLOGIN!=0)
    {
        printf("Eroare. Sunteti logat deja.\n");
        return;
    }

    mySendHeaderedMessage(fd,5,"login");

    char user[100],pass[100];
    printf("user: ");
    fflush(stdout);
    scanf("%s",user);
    printf("pass: ");
    fflush(stdout);
    scanf("%s",pass);
    mySendHeaderedMessage(fd,strlen(user),user);
    mySendHeaderedMessage(fd,strlen(pass),pass);

    char type=myRecvType(fd);
    int length=myRecvLength(fd);
    char *message=malloc(length+1);
    myRecvMessage(fd,length,message);
    message[length]='\0';
    printf("%s",message);
```

```
        fflush(stdout);
        free(message);

        if(type=='0')
        {
            flagLOGIN=1;
            strcpy(userGlobal,user);
        }
    }
```

5 Concluzii

În concluzie, soluţia propusă oferă funcţionalitatea cerută. Desigur, aceasta poate fi îmbunătăţită prin adăugarea unui coş de cumpărături, combinarea mai multor metode de recomandare (folosirea filtrării bazate pe produs dacă baza de date devine prea mare), realizarea unei interfeţe grafice.

References

1. Sabin Buraga şi Gabriel Ciobanu, Atelier de programare în reţele de calculatoare, Polirom, Iaşi, 2001.
2. Cursurile disciplinei *Reţele de calculatoare* din cadrul FII, UAIC (predate de Conf. dr. Alboaie Lenuţa)
3. Stevens, Fenner, Rudoff, UNIX Network Programming Volume 1, Third Edition: The Sockets Networking API
4. http://www.slideserve.com/nyoko/sisteme-de-recomandare-cu-filtrare-colaborativa
5. http://www.greenend.org.uk/rjk/tech/smtpreplies.html#connect
6. http://stackoverflow.com/questions/4813890/sqlite-or-mysql-how-to-decide
7. http://guidetodatamining.com/chapter2/
8. http://www.slideshare.net/lmsasu/curs-2-data-mining
9. https://en.wikipedia.org/wiki/Cosine_similarity
10. http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf
11. http://stackoverflow.com/questions/26849705/how-to-transfer-pdf-mp3-or-mp4-files-using-socket
12. http://liuj.fcu.edu.tw/net_pg/socket.html
13. http://zetcode.com/db/mysqlc/
14. http://www.codeproject.com/Articles/12893/TCP-IP-Chat-Application-Using-C