OPTIMIZACIJA PRETRAŽIVANJA BAZE PODATAKA U SISTEMU KOJI RADI U REALNOM VREMENU

Miloš Jevtić, Miroslav Petrović, Nikola Zogović, Milovan Stamatović, *Institut Mihajlo Pupin, Beograd*

Sadržaj – *U radu se opisuje tehnika koja znatno ubrzava pretraživanje baze podataka, što je čini pogodnom za primenu u sistemima koji rade u realnom vremenu. Tehnika je primenjena u softverskom modulu za identifikaciju izvora signala u automatizovanom sistemu za elektronsko izviđanje.*

1. UVOD

Kod sistema koji rade u realnom vremenu (*Real-time* - RT) vremenski interval od pojave pobude na ulazu sistema do generisanja odziva na tu pobudu je ograničen [1]. Da bi se taj uslov ispoštovao i poboljšale performanse sistema, potrebno je optimizovati sve operacije koje su uključene u generisanje odziva. U ovom radu se opisuje tehnika optimizacije pretraživanja baze podataka, kao jedne od često prisutnih operacija u radu RT sistema.

Pretraživanje baze podataka možemo ubrzati na dva nivoa [2]. Prvi nivo ubrzavanja je direktna posledica činjenice da je baza podataka fajl (ili skup fajlova) smešten na spoljašnjem memorijskom medijumu (najčešće hard disku). Uprkos razvoju brzih hard diskova i protokola za komunikaciju (SCSI, SATA), pristup operativnoj memoriji je još uvek značajno brži. Prema tome, ako podatke (ili deo podataka) iz baze podataka prebacimo u operativnu memoriju i tu vršimo pretraživanje, izbeći ćemo pristup disku i postići određeni stepen ubrzavanja. Ova tehnika je poznata kao keširanje podataka u RAM memoriji (RAM *caching*).

Prebacivanjem podataka u operativnu memoriju otvaramo put za drugi nivo ubrzavanja. Naime, mehanizmi (*engine*) koje DBMS (*Database Management System*) koristi za pretraživanje su prilagođeni širokom spektru tipova podataka i vrsta upita, pa ne mogu biti optimalni za svaki pojedinačni slučaj. Ako podatke prebacimo u operativnu memoriju, dobijamo potpunu kontrolu nad njihovom organizacijom i algoritmom pretraživanja, koji se sada mogu odabrati tako da budu optimalni za konkretan slučaj.

Jedan primer RT sistema u okviru kojeg se vrši pretraživanje baze podataka je sistem za elektronsko izviđanje (EI). Osnovne funkcije sistema za EI su uočavanje radara, komunikacionih uređaja i drugih izvora elektromagnetskog zračenja, određivanje karakteristika zračenja, kao i utvrđivanje položaja ovih izvora. Veoma važna funkcija sistema za EI je identifikacija izvora zračenja, na osnovu karakteristika signala koje izvor emituje.

Da bi se identifikovao izvor zračenja, potrebno je obradom signala primljenog od izvora odrediti karakteristične parametre signala i uporediti ih sa parametrima signala iz poznatih izvora. Parametre poznatih signala je pogodno čuvati u bazi podataka, pa otud potreba za pretraživanjem baze podataka u sistemu za EI.

Svaka promena u delu spektra koji pokriva sistem za EI predstavlja pobudu na ulazu sistema. Odziv sistema na ovakvu pobudu je potpuno obrađena informacija o izvoru zračenja, spremna za prikaz na grafičkom korisničkom interfejsu i eventualnu distribuciju putem komunikacionih kanala. Da bi sistem mogao da isprati sve promene u delu spektra koji je od interesa, vreme obrade mora biti ograničeno, pa stoga sistem za EI spada u kategoriju sistema koji rade u realnom vremenu.

2. IMPLEMENTACIJA

Implementacija tehnike za ubrzano pretraživanje baze podataka biće prikazana na primeru softverskog modula za identifikaciju izvora signala (*identification module* - IM) koji je deo sistemskog softvera automatizovanog sistema za EI. Najpre će biti izneti funkcionalni zahtevi koje IM treba da ispuni, a zatim i detalji implementacije.

Obradom detektovanog signala određuju se karakteristični parametri potrebni za identifikaciju: *tip* signala (celobrojna veličina koja može imati vrednost od 0 do M-1, gde je M broj različitih tipova signala koje sistem može da prepozna) i konačan broj parametara par_1, par_2, ..., par_N (realne veličine jednostruke preciznosti). Tip signala predstavlja tip modulacije, a parametri par_i su noseća frekvencija, širina dijagrama zračenja predajne antene itd. Parametri par_i su najčešće promenljivi i uzimaju vrednosti iz opsega [par_{imin}, par_{imax}]. Zbog toga se zapis u bazi podataka, koja se koristi za identifikaciju, sastoji od polja: *tip*, par_{1min}, par_{1max}, par_{2min}, par_{2max}, ... , par_{Nmin}, par_{Nmax}. Zapis sadrži i polje *ime*, koje predstavlja alfanumeričku identifikacionu oznaku izvora zračenja. Identifikacija je uspešna ako se pretraživanjem baze podataka pronađe zapis čije se polje *tip* slaže sa tipom detektovanog signala i pri tom važi: $par_i \in [par_{imin}, par_{imax}]$, za $i = 1, 2, ..., N$. Rezultat identifikacije je sadržaj polja *ime* datog zapisa.

Osnovna funkcija IM-a je identifikacija izvora signala prethodno pomenutim postupkom. Zahtevi za identifikacijom se javljaju vrlo često (sa svakom promenom parametara signala koje sistem prati), pa je akcenat stavljen na brzo pretraživanje baze podataka. Kada sistem detektuje izvor signala koji ne postoji u bazi podataka, operateru treba omogućiti dodeljivanje alfanumeričke oznake tom izvoru signala i formiranje novog zapisa u bazi podataka. Ova funkcionalnost takođe treba da bude ostvarena kroz IM.

Pre iznošenja detalja implementacije, treba napomenuti da je softverski sistem čiji je IM deo predviđen za rad pod operativnim sistemom zasnovanim na *Windows 2000* tehnologiji i realizovan korišćenjem programskog jezika *Visual* C++ i MFC *framework*a. Sistem je *multithreaded* i zahtevi IM-u upućivaće se konkurentno iz više niti. DBMS koji je upotrebljen je *Microsoft Access*.

Prva odluka koja je doneta je da se IM realizuje kao DLL. Na taj način je omogućeno da se implementacija IM-a može menjati bez potrebe da se glavna aplikacija ponovo povezuje [3]. Interfejs IM-a prema glavnoj aplikaciji je realizovan u duhu objektno orijentisanog programiranja, kao *singleton* [4] klasa koja ima javne metode za inicijalizaciju, deinicijalizaciju, pretraživanje postojećih zapisa i dodavanje novog zapisa. Ove metode su zaštićene od konkurentnog pristupa pomoću MFC podrške za kritične sekcije. Da bi interfejs ostao nezavisan od implementacije, primenjen je projektni uzorak most (*bridge*) [4].

Što se tiče načina pristupa bazi podataka, MFC *framework* nudi klase koje enkapsuliraju ODBC API, kao i

klase koje enkapsuliraju DAO COM interfejse. Imajući u vidu korišćeni DBMS, logičan izbor je DAO (prvenstveno namenjen za pristupanje lokalnim *Microsoft Access* bazama podataka [5]). Međutim, DAO nije pogodan za višenitne aplikacije jer se svi DAO pozivi moraju obaviti iz primarne niti procesa [6], pa je odabran ODBC.

Da bi se obezbedilo dovoljno brzo pretraživanje baze podataka, u realizaciji IM-a je primenjen prvi nivo ubrzavanja, tj. svi zapisi koji se koriste za identifikaciju su prebačeni u operativnu memoriju. Drugi nivo ubrzavanja nije primenjen iz dva razloga. Pre svega, struktura zapisa i postupak pretraživanja koji se koriste za identifikaciju su prilično kompleksni, pa se pretraživanje ne može optimizovati primenom neke od standardnih metoda kao što je binarno pretraživanje ili primena balansiranog stabla ili *hash* tabele. Pored toga, moguće je da se kod dva ili više zapisa koji se slažu po tipu signala opsezi parametara par_i delimično preklapaju, tako da se može dogoditi da rezultat pretraživanja ne bude jedinstven. Pošto se ne može primeniti neka optimalna metoda pretraživanja, primenjena je najjednostavnija – sekvencijalno pretraživanje.

Prilikom svake promene u kolekciji zapisa, vrši se provera da li postoje preklapanja opsega i informacija o tome se pamti. Ako nema preklapanja, rezultat pretraživanja je jedinstven i pretraživanje se prekida čim se naiđe na zapis koji zadovoljava upit. Ako preklapanja postoje, rezultat pretraživanja ne mora biti jedinstven, pa se uvek pretražuje cela kolekcija. Na zahtev glavne aplikacije za pretraživanjem kolekcije zapisa, IM vraća listu identifikacionih oznaka koja može biti prazna ako nijedan zapis ne zadovoljava upit.

Dodavanje novog zapisa podrazumeva dodavanje u bazu podataka i u kolekciju zapisa u operativnoj memoriji (što zahteva alociranje dodatnog memorijskog prostora).

Zbog sekvencijalnog pretraživanja, logično je kolekciju zapisa čuvati u STL *container*u *list* ili *vector*. Što se tiče brzine iteracije, *vector* je efikasniji. Ako se posmatra vreme potrebno za dodavanje novog zapisa pogodnija je lista, jer je uvek potrebno alocirati memoriju samo za jedan zapis i izvršiti jedno kopiranje. Kod *vector*a čiji je kapacitet N, a koji već sadrži N zapisa, dodavanje novog zapisa podrazumeva realokaciju memorije i N+1 kopiranja.

Pri projektovanju IM-a procenjeno je da će se dodavanje novih zapisa događati relativno retko, pa je za čuvanje kolekcije zapisa odabran *vector*, pri čemu se kod inicijalne alokacije (i eventualnih kasnijih realokacija) rezerviše kapacitet za 5 do 10 procenata veći od neophodnog (čime se dodatno smanjuje učestanost realokacije).

3. REZULTATI

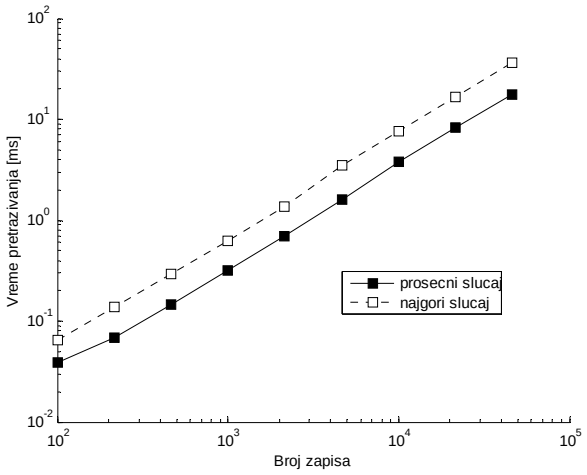
Da bi se utvrdile performanse IM-a i primenjene tehnike za ubrzavanje pretraživanja baze podataka, izvršena su dva testa. Za merenje vremena korišćen je sistemski brojač visoke rezolucije poznat kao *performance counter*. Testovi su obavljeni na PC kompatibilnom računaru sa sledećom konfiguracijom: procesor *Intel Celeron* (800 MHz), memorija 512 MB SDR (100 MHz), operativni sistem *Windows XP Professional*.

U prvom testu je ispitana zavisnost vremena pretraživanja od broja zapisa u bazi podataka, korišćenjem IM-a. Posmatrana su dva slučaja: prosečan i najgori. Prosečan slučaj podrazumeva idealnu situaciju u kojoj nema preklapanja opsega. Za svaki zapis je izmereno vreme potrebno za njegovo pronalaženje (svi zapisi u bazi podataka su poznati), a srednja vrednost izmerenih vremena predstavlja vreme pretraživanja u prosečnom slučaju. Najgori slučaj podrazumeva da preklapanja opsega postoje. Vreme pretraživanja praktično ne zavisi od parametara pretraživanja jer se u svakom slučaju prolazi kroz sve zapise.

Rezultati prvog testa su prikazani u tabeli 1 i u skladu su sa očekivanjima. Vreme pretraživanja praktično linearno raste sa brojem zapisa, što je, sa obzirom na činjenicu da je pretraživanje sekvencijalno, sasvim logično.

Tabela 1. *Zavisnost vremena pretraživanja od broja zapisa*

Broj zapisa	Vreme pretraživanja (prosečan slučaj) [ms]	Vreme pretraživanja (najgori slučaj) [ms]
100	0.038945	0.064799
215	0.069128	0.137531
464	0.147172	0.290135
1000	0.321334	0.619380
2150	0.686144	1.373107
4640	1.607921	3.534220
10000	3.768964	7.735438
21500	8.381897	16.581716
46400	17.725084	36.207311



Sl. 1. *Zavisnost vremena pretraživanja od broja zapisa*

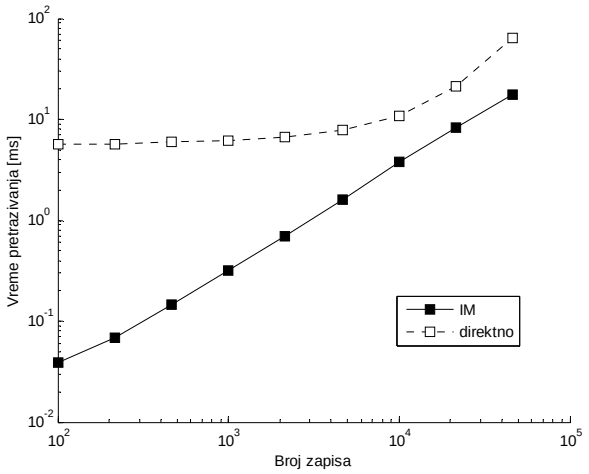
Na Sl. 1 su grafički prikazani rezultati prvog testa. Zbog velikog raspona broja zapisa, svi grafici su u log-log razmeri.

U drugom testu je izvršeno poređenje vremena koje je potrebno za pretraživanje pomoću IM-a sa vremenom koje je potrebno za direktno pretraživanje baze podataka. Za potrebe ovog testa napravljen je DLL koji ima isti interfejs kao IM, ali ne prebacuje zapise u operativnu memoriju, već pretraživanje baze podataka vrši direktnim postavljanjem SQL upita. Testiranje ovog DLL-a je obavljeno za prosečan slučaj, a rezultati poređenja sa IM-om su prikazani u tabeli 2. U istoj tabeli je dat i odnos vremena direktnog pretraživanja i vremena pretraživanja pomoću IM-a. Rezultati su grafički prikazani na Sl. 2 i 3.

Kao što se očekivalo, IM je brži od DLL-a sa direktnim pretraživanjem. Ova prednost je naročito izražena kod baza podataka koje imaju do 2000 zapisa.

Tabela 2. *Rezultati poređenja (prosečan slučaj)*

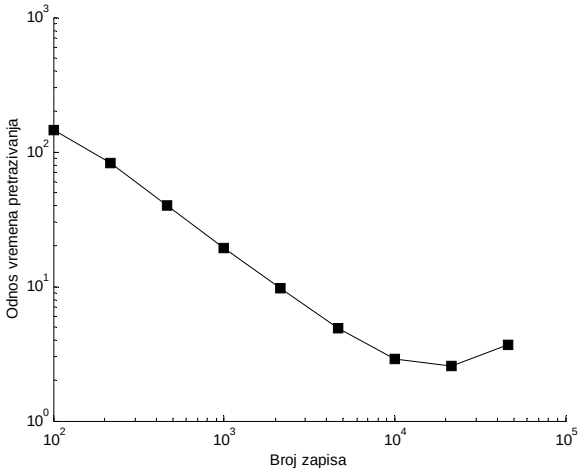
Broj zapisa	Vreme pretraživanja (IM) [ms]	Vreme pretraživanja (direktno) [ms]	Odnos direktno/IM
100	0.038945	5.655547	145.2188
215	0.069128	5.755658	83.2609
464	0.147172	5.916062	40.1983
1000	0.321334	6.185168	19.2484
2150	0.686144	6.648004	9.6889
4640	1.607921	7.895726	4.9105
10000	3.768964	10.793454	2.8638
21500	8.381897	21.461152	2.5604
46400	17.725084	64.624157	3.6459



Sl. 2. *Rezultati poređenja (prosečan slučaj)*

4. ZAKLJUČAK

U radu je pokazano kako se primenom jednostavne tehnike, uz male troškove razvoja, može značajno ubrzati pretraživanje baze podataka. Mana prikazane tehnike je zauzeće operativne memorije, koje zavisno od veličine i broja zapisa može biti značajno. Zbog toga odluku o primeni pomenute tehnike treba doneti uzimajući u obzir količinu podataka, količinu raspoložive memorije i potrebne performanse sistema.



Sl. 3. *Odnos vremena pretraživanja*

U konkretnom slučaju automatizovanog sistema za EI, struktura zapisa i upit su takvi da primena nijedne od standardnih tehnika za optimizaciju pretraživanja ne bi doprinela poboljšanju performansi. Ali, već sama primena RAM *caching* tehnologije značajno je poboljšala odziv sistema.

Buduće istraživanje biće usmereno na pokušaj pronalaženja specifične tehnike optimizacije, koja će dodatno ubrzati identifikaciju izvora signala i omogućiti da IM isprati razvoj ostalih delova sistema.

LITERATURA

[1] P. Lapplante, *Real-time Systems, Design and Analyses*, IEEE Press, New York, 1997.

[2] M. Jevtić, *Softverski modul za pretraživanje baze podataka u realnom vremenu*, diplomski rad, 2003.

[3] J. Richter, *Programming Applications for Microsoft Windows*, Microsoft Press, 1999.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[5] M. Pietrek, "Under The Hood", *Microsoft Systems Journal*, June 1999.

[6] D. J. Kruglinski, S. Wingo, G. Shepherd, *Programming Microsoft Visual C++*, Microsoft Press, 1998.

Abstract – The paper describes a techique that accelerates database search, and is suitable for application in real-time systems. Technique is applied in emitter identification software module used in automated electronic intelligence system.

DATABASE SEARCH OPTIMIZATION IN REAL-TIME SYSTEMS

Miloš Jevtić, Miroslav Petrović,
Nikola Zogović, Milovan Stamatović