SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Sustavi za rješavanje problema pristupa podacima

Alen Rakipović
Voditelj: *Mile Šikić*

Zagreb, lipanj 2011.

SADRŽAJ

1. Uvod

Izrada web aplikacija je jedna od najproduktivnijih grana računarstva. Uobičajena arhitektura web aplikacija je klijent - poslužitelj arhitektura. Najčešće, aplikacija dohvaća podatke iz baze podataka i prosljeđuje ih na korisničko sučelje. Budući da se dostupnost Interneta povećava iz dana u dan, aplikacije moraju omogućavati istovremeni pristup milijunima korisnika. Ovdje često dolazi do problema zagušenja zbog prevelikog broja pristupa bazi podataka.

Navedni problem je moguće uspješno riješiti na nekoliko načina. U ovom radu će biti predstavljena dva načina rješavanja problema. Prvi je sustavima za raspodijeljeno čuvanje podataka u memoriji. *Memcached* je sustav visokih preformansi koji služi za pohranjivanje objekata u raspodijeljenoj memoriji. Iako mu je područje primjene vrlo općenito, najčešće se koristi za ubrzavanje dinamičkih web aplikacija ublažavanjem opterećenja na baze podataka. Drugi sličan sustav je *Redis* koji je zapravo sustav za napredno mapiranje (ključ-vrijednost) podataka. Iduće moguće rješenje je baza podataka koja je spremljena u radnu memoriju računala. U svrhu ovog rada isprobana su dva pristupa: vlastita baza podataka i gotovo rješenje (*HSQLDB*).

Drugo poglavlje donosi opis aplikacije koja računa zastupljenost metala u proteinima. U trećem poglavlju su opisani sustavi *Memcached* i *Redis* te su navedene neke njihove primjene te pokušaj korištenja na konkretnoj aplikaciji. Iduće poglavlje opisuje baze podataka u memoriji i pokušaj ubrzavanja aplikacije korištenjem istih. Nakon toga slijedi zaključak.

2. Web aplikacija

Web aplikacija koju se pokušalo ubrzati je razvijena prilikom izrade magistarskog rada Peretin (2010.). Aplikacija ima za cilj omogućiti pregled statističkih podataka o prisutnosti pojedinih metala u proteinima, kao i učestalosti pojavljivanja više metala u nekom proteinu. Također, ostvarena je analiza podataka o prisutnosti metala u proteinima, njihovom koordinacijskom broju i aminokiselinskim ostatcima te ligandima na koje se vežu. Aplikacija se bazira na klijent - poslužitelj arhitekturi gdje se za odabrane metale postavlja četiri upita prema bazi podataka i računa osam statistika na poslužitelju. Pri tome nastaju problemi s brzinom obrade podataka jer je broj upita često jako velik. Ovakve probleme je moguće riješiti različitim sustavima i tehnikama za pohranu podataka koje su opisane u idućim poglavljima.

3. Memcached i Redis

Ovo su sustavi visokih preformansi koji služe za pohranjivanje objekata u raspodijeljenoj memoriji. Iako su im područja primjene vrlo općenita, najčešće se koriste za ubrzavanje dinamičkih web aplikacija ublažavanjem opterećenja na baze podataka.

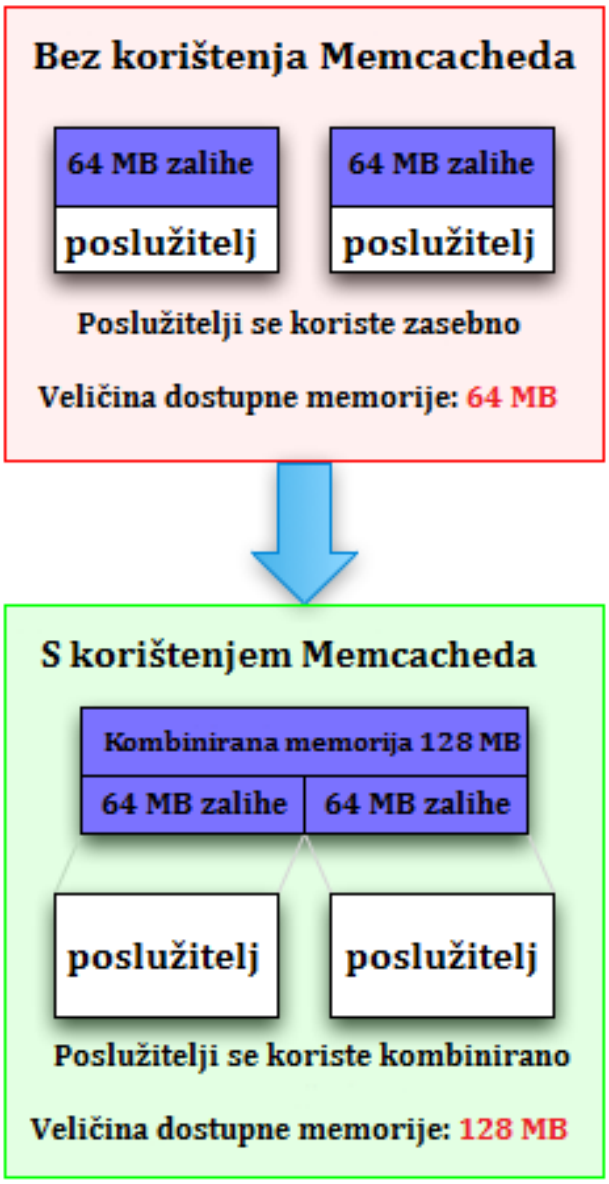
3.1. Memcached

Memcached je sustav koji je Brad Fitzpatrick razvio za svoju web stranicu *LiveJournal* 2003. godine. On pruža ogromnu hash tablicu raspodjeljenu na više računala. Kada se tablica napuni podacima, umetanje novih se izvršava tako da se prvo obriše najstariji zapis u tablici i na njegovo mjesto se upiše novi podatak. Aplikacije koje koriste ovaj sustav najčešće imaju nekakvu bazu podataka te *Memcached* čini poveznicu između baze podataka i aplikacije. Neke od stranica koje koriste ovaj sustav su *Youtube, Facebook, Twitter, Zynga* i druge.

3.1.1. Arhitektura

Sustav koristi klijent - poslužitelj arhitekturu. Poslužitelj održava asocijativno polje koje klijent popunjava i nad kojim radi upite.

Klijenti koriste funkcije iz klijentskih biblioteka za pristup poslužitelju koji obično budu dostupni na priključnici 11211. Svaki klijent zna svakog poslužitelja, budući da poslužitelji ne komuniciraju međusobno. U slučaju da klijent želi pročitati ili zapisati podatak s odgovarajućim ključem, on prvo poziva funkciju koja računa sažetak ključa da bi odredio na kojem se poslužitelju obavlja akcija. Poslužitelj čuva podatke u svojoj radnoj memoriji. U slučaju da se ta memorija prepuni, briše se najstariji zapis te zbog toga klijenti ne smiju općenito pretpostaviti da je podatak tamo gdje bi trebao biti spremljen. Ovakav način rada je definiran Memcached-protokolom.



Slika 3.1: Usporedba dostupnosti memorijskog prostora s i bez korištenja Memcacheda

Najčešće svaki klijent ima jednaku funkciju za računanje sažetka kako bi svi klijenti imali mogućnost čitati podatke međusobno. Tipičan razvoj aplikacije koja koristi ovaj sustav podrazumijeva veći broj klijenata i poslužitelja, međutim moguće je koristiti sustav i na samo jednom računalu.

3.1.2. Sigurnost

Većina aplikacija koje koriste sustav su unutar sigurne mreže gdje se svaki klijent može spojiti na svaki poslužitelj. Ipak, postoje slučajevi gdje mreža nije osigurana pa je zbog

toga razvijena *SASL* autentifikacija koja zahtjeva binarni protokol.

3.1.3. Primjer koda

U nastavku je prikazan primjer funkcije dohvaćanja iz baze bez korištenja sustava *Memcached*.

```
funkcija primjer_dohvacanja_iz_baze(int protein_id)
    rezultat = upit_baza("SELECT * FROM proteini WHERE
                        proteini.id = ?", protein_id);
    vrati rezultat;
```

Za svaki podatak koji želimo dohvatiti potrebno je poslati upit na bazu podataka što ponekad nije dovoljno brzo.

Slijedi pseudokod funkcije iznad, ali sada uz korištenje sustava *Memcached*.

```
funkcija primjer_dohvacanja_iz_baze_memcached(int protein_id)
    rezultat = upit_memcached(protein_id);
    ako (rezultat_ne_postoji)
        rezultat = upit("SELECT * FROM proteini WHERE
                        proteini.id = ?", protein_id);
        spremi_memcached(protein_id, rezultat);

    vrati rezultat;
```

Prvo se izvršava upit na sustav *Memcached* (koji je puno brži od pristupa bazi podataka), te ukoliko nema podatka pod ključem (protein_id) rezultat se dohvaća iz baze podatka i sprema u tablicu *Memcacheda*.

3.2. Redis

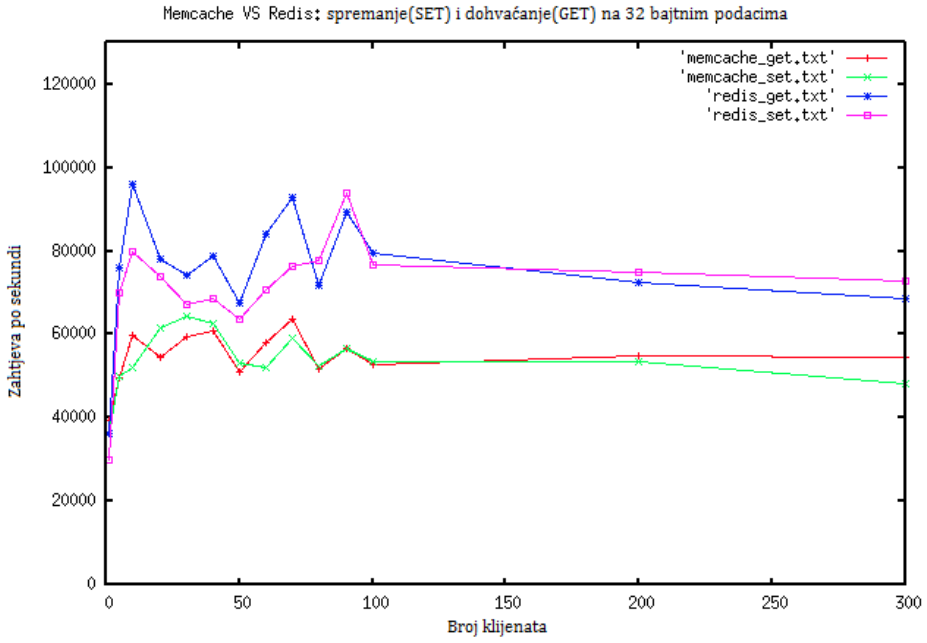
Redis je sustav za pohranjivanje podataka u obliku ključ - podatak. Prednost mu je u velikom broju klijenata za razne programske jezike te tome što za razliku od ostalih sličnih sustava ključ ne mora biti niz znakova. Kod ovog sustava ključevi osim znakovnih nizova mogu biti liste znakovnih nizova, skupovi znakovnih nizova, poredani skupovi znakovnih nizova te sažetci gdje su ključevi znakovni nizovi, a vrijednosti znakovni nizovi ili cjelobrojne vrijednosti. *Redis* u velikom broju slučajeva podatke čuva u radnoj memoriji računala, no može biti podešen da sprema podatke u virtualnu memoriju. Perzistencija podataka je ostvarena na dva načina: jedan je stvaranje

slike podataka i asinkrono prenošenje u memoriju s vremena na vrijeme, a drugi je izravno prenošenje svih podataka u memoriju. *Redis* se koristi na sličan načina kao i *Memcached* te će ovdje biti preskočen primjer pseudokoda.

3.3. Usporedba učinka

Usporedba učinka ovakvih sustava se može raditi na temelju mnogih parametara kao što su brzina, skalabilnost, pouzdanost i druge. Mjerenje sirove brzine je ograničenje upita po watu. Energija je veliki problem, pa u slučaju korištenja više kompjutera za opsluživanje nekog broja zahtjeva u sekundi donosi i veći cijenu. Uz to, velike sustave je teško održavati, a i velika većina web inovacija dolazi od malih tvrtki koje nemaju velike sustave. Brzina je važna čak i kada postoji skalabilan sustav s puno računala jer je i tada bitno iskoristiti sav potencijal tih računala.

Usporedba ovih dvaju sustava nije nimalo trivijalan posao jer uz to što *Redis* pruža više mogućnosti od *Memcacheda*, postoje razne implementacije klijenata za svaki programski jezik, te se treba paziti na okolinu testiranja, broj klijenata i slično. U nastavku su dani rezultati za klijent u programskom jeziku C, testirana su dohvaćanja i spremanja u memoriju u oba sustava na većem broju klijenata s veličinom podataka od 32 bajta. Rezultati su preuzeti s web stranice [1].



Slika 3.2: Usporedba učinka Redis i Memcached sustava

[1] `http://antirez.com/post/redis-memcached-benchmark.html`

Iz grafa se može primijetiti da nema pretjerane razlike između učinka ovih dvaju sustava te da je sustav *Redis* u ovom testu malo brži. Međutim, ovo generalno ne mora biti slučaj, već sve ovisi o konkretnoj situaciji u kojoj se sustavi upotrebljavaju.

Općenito, da bi se zaista upoznalo mogućnosti sustava za raspodijeljeno čuvanje podataka u memoriji, treba proučiti jedan drugi aspekt. *Redis* može izvoditi složene atomičke operacije nad kompleksnim strukturama podataka kao što su liste, skupovi, sortirani skupovi i slično, istom brzinom kao što se izvode operacije spremanja(SET) i dohvaćanja(GET) iz memorije. Ovo daje puno mogućnosti u samo jednoj operaciji i moglo bi pružiti dobre rezultate po pitanju učinka.

Navedeni sustavi daju izvrsne rezultate kada se kombiniraju s aplikacijama koje se izvode na velikom broju poslužitelja te zahtjevaju veliku količinu memorije koja treba biti na raspolaganju klijentima. U tom slučaju memorijska ušteda je reda veličine broja poslužitelja, dok je vremenska i veća. Budući da je se ova aplikacija izvodi na jednom poslužitelju i ne zahtjeva veliku količinu podataka - baza je veličine nekoliko desetaka MB, razmatrano je pohranjivanje rezultata svih mogućih kombinacija upita na bazu. Ovaj način nije dao zadovoljavajuće rezultate jer je broj mogućih kombinacija reda veličine 35! što zahtjeva ogromnu veličinu radne memorije koja nije na raspolaganju jer se, kako je navedeno iznad, aplikacija izvršava na jednom poslužitelju.

Pozitivne strane sustava za raspodjeljeno čuvanje podataka u memoriji su vremensko i memorijsko poboljšanje složenosti u web aplikacijama. Preporuča se njihova uporaba u kombinaciji s aplikacijama koje se izvode na puno poslužitelja i kod kojih je veliki problem s brzinom dohvaćanja velike količine podataka. Negativni efekti korištenja ovakvih sustava su nepouzdanost iz razloga što se sve pohranjuje u radnoj memoriji pa se u slučaju gašenja ili kvara sustava sve briše iz memorije, te nemaju mogućnost ostvarivanja sigurnosne kopije podataka.

4. Baza podataka u radnoj memoriji

Ubrzanje aplikacija može se postići i prebacivanjem podataka u radnu memoriju gdje je brzina pristupa puno veća nego brzina pristupa memoriji tvrdog diska. Ovdje ulaze baze podataka koje se primarno oslanjaju na pohranu podataka u radnoj memoriji računala. Pristupanje podacima u radnoj memoriji je brže jer smanjuje ulazno/izlazne aktivnosti što omogućava brže i lakše predvidivo ponašanje od tvrdog diska. Ovo rješenje se čini idealno za aplikacije kod kojih se ne koristi više poslužitelja, te nisu memorijski zahtjevne pa se svi podaci mogu prebaciti u radnu memoriju računala.

4.1. ACID

ACID je skup značajki koji osiguravaju da se transakcije na bazi podataka izvrše pouzdano. Transakcija je jedna logička operacija nad podacima.

- atomarnost zahtjeva da modifikacije nad podacima moraju pratiti "Ništa ili sve" pravilo. Svaka transakcija mora biti atomarna, tj. ako se bilo koji dio transakcije ne izvrši, cijela transakcija se ne izvršava i podaci ostaju nepromijenjeni.
- konzistentnost osigurava potpuno povjerenje u podatke koji se nalaze u bazi, tj. da će bilo kojom transakcijom baza iz konzistentnog stanja ući u drugo konzistentno stanje.
- izolacija se odnosi na zahtjev da samo jedna operacija može pristupiti određenom podatku u određenom vremenu.
- postojanost je svojstvo koje omogućava bazi podataka da se oporavi od bilo kakve greške koja se dogodi tijekom transakcije.

Kod ove vrste baza zadovoljena su prva tri svojstva, dok se postojanost pokušava zadovoljiti na način da se stvori slika stanja baze u određenom trenutku. Ovo se ostvaruje periodički ili pri kontroliranom gašenju baze. Međutim, uz stvaranje slike potrebno je ostvariti i jedan od sljedećih mehanizama kako bi se ostvarila postojanost:

- logiranje transakcija je pohranjivanje promjena kroz koje je baza prošla.

– pohranjivanje podataka u nepromijenjivu radnu memoriju koja predstavlja statičku radnu memoriju podupiranu s baterijom.

– implementacije baze tako da se ona lako može replicirati, a samim time i lagano zamijeniti u slučaju kvara orginalne.

Navedeni postupci se odnose na razne mogućnosti rušenja sustava kao što je prekid dovoda električne energije ili greška u radu operacijskog sustava.

4.2. Vlastita baza

Ostvarenje vlastite baze korištene za potrebe ove aplikacije se sastoji od 13 razreda implementiranih u programskom jeziku Java. Svaka tablica u bazi podataka prikazana je razredom odgovarajućeg imena gdje su redovi pohranjeni u listu. Svaki red je modeliran zasebnim razredom s članskim varijablama koje odgovaraju atributima tablice. Uz navedene razrede, implementiran je još razred koji sadrži metode za stvaranje tablica u memoriji te njihovo spremanje na tvrdi disk. Ovaj razred je modeliran kao Singleton oblikovni obrazac da bi se spriječilo uvišestručavanje podataka.

Rezultati ovakve implementacije nisu pokazali pretjeranog uspjeha ponajviše zbog nemogućnosti efikasnog izvršavanja spajanja nekoliko tablica koje je nužno za računanje pojedinih statistika.

4.3. HyperSQLDB

HyperSQL baza podataka je relacijski sustav za upravljanje bazom podataka napisan u programskom jeziku Java. Sadržava JDBC standard i podržava velik dio SQL standarda. Omogućuje brzi i prostorno vrlo mali sustav koji podržava pohranjivanje podataka na tvrdi disk i u radnu memoriju računala. Također, podržani načini rada su serverski i ugradbeni.

Dodatni alati koji dolaze s ovim sustavom su minimalni web poslužitelj, komandno i grafičko sučelje, te nekoliko demonstacijskih primjera.

HSQLDB ima dva glavna tipa rada kako bi se očuvala postojanost podataka. Prvi način je spremanje svih modifikacija nad podacima u obliku SQL skripte koja se izvršava kada sustav započinje s radom. Tada se svi podaci rekonstruiraju u memoriji računala. Ovaj tip rada nije pogodan za tablice koje zahtjevaju mnogo memorijskog prostora, ali pruža poprilično dobre učinke i lako je otkriti grešku u podacima ako ona

postoji. Drugi tip omogućava pohranjivanje gigabajta podataka, ali pod cijenu smanjenja brzine izvođenja. Sustav dohvaća samo dio podataka i obavlja sinkronizaciju na zaključenju transakcije. Budući da sustav uvijek dohvaća sve retke koji su zahvaćeni transakcijom ovaj postupak zahtjeva da se posao podijeli na manje dijelove. Arhitektura ovog sustava je podijeljena na dijelove gdje svaki dio čini jedna datoteka koja se smješta na mjesto na tvrdom disku odakle se pozvala izvršna datoteka. Glavni dijelovi su *test.data* koji čine podaci zapisani u tablice, *test.properties* u kojem su zapisane razne postavke baze, *test.script* u kojoj se čuvaju podaci o transakcijama. Uz to, u nekim slučajevima postoje još tri datoteke: *test.backup* u kojoj je sačuvana slika podataka u nekom vremenskom trenutku, zatim *server.properties* u kojoj su zapisani podaci o poslužitelju i *webserver.properties* u kojoj je zapisana konfiguracija web poslužitelja.

Sama ugradnja ovakve baze podataka u web aplikaciju je zahtjevala implementaciju dodatnih razreda za kopiranje podataka iz osnovne baze u memorijsku jer je trebalo prilagoditi SQL zapise dviju baza. Ovaj način također nije proizveo željeno ubrzanje zbog nedostatka mogućnosti memorijske baze kao što su neke ugrađene funkcije i problematični prijenos podataka te veliki broj spajanja tablica. Ovo rješenje će se pokušati optimirati u budućnosti.

Karakteristika	Poslužitelja	Memcached/Redis	Baza u memoriji
memorija	jedan	Aplikacija ima na raspolaganju memorije koliko ima poslužitelj.	Aplikacija ima na raspolaganju memorije koliko ima poslužitelj.
	više	Aplikacija ima na raspolaganju memorije **koliko svi poslužitelji kombinirano**.	Aplikacija ima na raspolaganju memorije koliko poslužitelj na kojem se nalazi baza.
brzina	jedan	Brzina aplikacije ovisi o brzini poslužitelja.	Brzina aplikacije ovisi o brzini poslužitelja.
	više	Aplikaciji se brzina povećava proporcionalno. **broju poslužitelja**.	Brzina aplikacije ovisi o brzini poslužitelja.
koristiti za	jedan	Aplikacije s malo korisnika i velikim brojem ponavljanja upita.	Aplikacije s malo korisnika i neponavljanjem upita.
	više	Aplikacije s puno korisnika i velikim brojem ponavljanja upita.	Nije dobro rješenje za aplikacije s puno korisnika.

Tablica 4.1: Usporedba dviju navedenih metoda

5. Zaključak

Web aplikacije su jedno od žarišta računarstva. S uobičajenom klijent - poslužitelj arhitekturom najčešće je potrebno sačuvati nekakve podatke i obaviti manipulaciju nad njima. U ovom radu se koristi aplikacija koja omogućava pregled statističkih podataka o prisutnosti pojedinih metala u proteinima, kao i učestalosti pojavljivanja više metala u nekom proteinu. Za pohranu baze proteina se koristi MySQL baza podataka. Inače se za pohranu podataka koriste razne vrste baza. Međutim, sam pristup podacima zna biti izvor problema zbog svoje nedovoljne brzine. U ovom radu je predstavljeno nekoliko mogućih rješenja tog problema. Danas sve popularniji sustavi *Memcached* i *Redis* nude sponu između baze podataka i same aplikacije na način da spremaju rezultate upita u radnu memoriju računala te ih pri ponovnom dohvaćanju dohvaćaju iz radne memorije, a ne iz baze podataka što je značajna ušteda vremena. U konkrentom slučaju ovi sustavi se nisu pokazali dobrim rješenjem jer aplikacija pruža mogućnost generiranja velikog broja različitih upita koje nije moguće sve pohraniti u radnoj memoriji računala. Uz navedene sustave, dano je i rješenje s bazama podataka u memoriji. Koristila se gotova i novoizgrađena baza podataka. Kod novoizgrađene baze se javio problem nedovoljne brzine operacije spajanja tablica. U izboru gotove baze podataka, autor se odlučio na korištenje *HSQLDB* koja nudi najbolje rezultate od trenutno besplatno dostupnih baza podataka. Iako trenutno ubrzanje nije zadovoljavajuće, ovo ostaje najbolje moguće rješenje i nastavit će se ispitivanje i rad s ovim rješenjem.

Iako postoji puno različitih sustava koji nude puno mogućnosti za rješenje problema brzog pristupa i manipulaciji s bazom podataka, još uvijek postoje problemi za koje je potrebno ostvariti drugačije rješenje.

6. Literatura

Goran Peretin. Baza zastupljenosti metala u proteinima. Magistarski rad, FER, 2010.

The HSQL Development Group, Blaine Simpson, and Fred Toussi, HyperSQL User Guide, HyperSQL Database Engine (HSQLDB) 2.2, svibanj 2011.

Memcached documentation, Brad Fitzpatrick i ostali

Redis documentation, Salvatore Sanfilippo i Pieter Noordhuis