Matematički fakultet
Univerzitet u Beogradu

Sinhronizacija podataka u distribuiranim bazama podataka: ponovljeni podaci i lenjo ažuriranje

Master rad

Mentor:
Prof. dr. Gordana Pavlović-Lažetić

Autor:
Milica Janačkov

Beograd, 2011.

SADRŽAJ

1 O DISTRIBUIRANIM SISTEMIMA

Tehnologija distribuiranih sistema baza podataka predstavlja spoj dva dijametralno suprotna koncepta obrade podataka: sistema baza podataka i tehnologije računarskih mreža. Pojava sistema baza podataka dovela je do integracije podataka i omogućila centralizovano, a time i kontrolisano pristupanje i upravljanje podacima nasuprot dotadašnjim načinima obrade podataka gde je svaka aplikacija definisala i upravljala svojim skupom podataka. Tehnologija računarskih mreža, sa druge strane, promoviše i donosi decentralistički način rada u računarskom sistemu tako da se na prvi pogled može učiniti nemogućim da ova dva koncepta zajedno donesu tehnologiju koja omogućava napredniji način upravljanja podacima. Ono što je ključ u razumevanju koncepta distribuiranih sistema baza podataka je integracija kao i činjenica da je moguće postići integraciju bez centralizacije, a to je upravo ono što distribuirani sitemi baza podataka imaju za cilj.[1]

Termin distribuirana obrada (procesiranje) je teško precizno definisati, s obzirom da određeni stepen distubuirane obrade postoji u svakom računarskom sistemu čak i na jednoprocesorskim računarima gde su procesorska i funkcionalnost ulazno/izlaznih uređaja u nekim delovima preklopljene, a sami uređaji razdvojeni. Distribuirani računarski sistem može se definisati kao skup autonomnih računarskih elemenata (ne obavezno homogenih) koji su povezani u računarsku mrežu i koji su kooperativni u izvršavanju zadataka. [1]

Ključno pitanje je: Šta se sve distribuira? Iz prethodne definicije sledi da se može distribuirati logika (metoda) izvršenja zadatka, kao i podaci koji učestvuju u procesu. Dalje, različite funkcije računarskog sistema mogu biti dodeljene različitim delovima hardvera ili softvera u računarskom sistemu. Konačno, kontrola izvršenja procesa može biti distirbuirana umesto da se odvija na jednom računaru.

Još jedno bitno pitanje je: Zašto uopšte distribuiramo računarski sistem? Jednostavan odgovor je da distribuirana obrada više odgovara organizacionoj strukturi današnjih široko distribuiranih poslovnih preduzeća i da je distribuirani računarski sistem pouzdaniji i spremniji da odgovori na njihove zahteve. Što je još važnije, mnoge od postojećih aplikacija su po prirodi distribuirane, npr. veb aplikacije, e-biznis preko Interneta, multimedijalne aplikacije kao što su info-na-zahtev, sistemi za kontrolu prozivodnje itd. Globalno gledano, ključni razlog za distibuiranu obradu je potreba da se izađe na kraj sa velikim problemima u upravljanju podacima, sa kojima se susrećemo danas. Ukoliko je moguće razviti odgovarajući softver za distribuiranu obradu, komplikovani problemi se mogu razrešiti jednostavnom podelom na manje delove. Ovi delovi problema (i rešenja) dodeljuju se delovima softvera na odvojenim računarima čime se stvara sistem koji na odvojenim elementima efikasnije radi na izvršenju zadatka. Distribuirane sisteme baza podataka treba posmatrati i u ovom svetlu i tretirati ih kao pomoćno sredstvo koje može učiniti distribuiranu obradu lakšom i efikasnijom.

2 DISTRIBUIRANE BAZE PODATAKA

Distribuirana baza se može definisati kao skup više logički međuzavisnih baza podataka distribuiranih kroz računarsku mrežu. Sistem za upravljanje distribuiranim bazama podataka je programski sistem koji omogućava efikasno upravljanje distribuiranim bazama podataka kao i nevidjivost lokacije.[1]

Neformalno govoreći, distribuirana baza podataka je baza podataka koja se ne nalazi u celosti na jednoj fizičkoj lokaciji, već je razdeljena na više lokacija koje su povezane komunikacionom mrežom. Svaka lokacija (čvor komunikacione mreže) poseduje sopstveni, autonomni sistem za upravljanje bazama podataka sa sopstvenom kontrolom, upravljačem transakcijama, oporavkom od pada i ostalim značajnim funkcijama.[2]

Prednosti koje donosi distribuirani koncept su sledeće:

- Nevidljivo (eng. *transparent*) upravljanje distribuiranim sistemom
- Pouzdanost i raspoloživost
- Efikasnost i fleksibilnost
- Veći kapacitet i postupni rast
- Lokalna autonomija podataka, upravljanja i kontrole

2.1 Nevidljivo upravljanje distribuiranim sistemom

Nevidljivo (engl. *transparent*) upravljanje distribuiranim sistemom se odnosi na razdvajanje semantike sistema od problema implementacije sistema. Drugim rečima, sistem sakriva implementaciju od korisnika čime omogućava široku podršku za razvoj raznih kompleksnih aplikacija nezavisno od interne organizacije sistema i podataka u njemu.

Postoji nekoliko tipova nevidljivosti u distribuiranom sistemu koje treba obezbediti da bi se sistem mogao smatrati potpuno nevidljivim.

Nevidljivost podataka je osnovni oblik nevidljivosti distribuiranog sistema koji treba obezbediti, a odnosi se na nezavisnost korisničkog programa od promena u definiciji i organizaciji podataka unutar distribuiranog sistema. Ovo se odnosi na promene na nivou logičke strukture kao i na promene na nivou fizičke strukture i organizacije podataka.

U centralizovanom sistemu, jedini oblik nevidljivosti koji treba obezbediti odnosi se na podatke, dok u distribuiranom sistemu postoji još jedan aspekt na koji se nevidljivost može odnositi, a to je komunikaciona mreža. Postoje dva tipa nevidljivosti komunikacione mreže: nevidljivost lokacije, koja se odnosi na to da korisnik i njegov program ne treba da znaju na kojoj se lokaciji u sistemu nalaze podaci koji su im potrebni i nevidljivost imenovanja svakog objekta u bazi. Nevidljivost komunikacione mreže treba da obezbedi da distribuirani sistem za korisnika izgleda isto kao centralizovani.

Ukoliko su podaci u distribuiranom sistemu ponovljeni onda je nevidljivost ponavljanja podataka još jedan od zahteva koji je potrebno ispuniti, a podrazumeva sakrivanje činjenice o postojanju kopija od korisnika kao i održavanje konzistentnosti kopija od strane sistema bez potrebe da korisnik vodi računa o tome.

Sa druge strane, ukoliko su podaci particionirani, strategija obrade upita nad skupom podataka razdeljenim na više lokacija u distribuiranom sistemu takođe mora biti skrivena od korisnika i njegovog programa.

2.2 Pouzdanost i raspoloživost

Distribuirani sistemi mogu da nastave svoje funkcionisanje i kada neki od čvorova distribuiranog sistema iz nekog razloga postane nedostupan ili kada otkaže neka veza u komunikacionoj mreži, čime se povećava pouzdanost sistema kao i raspoloživost podataka. Čak i kada je nemoguće doći do nekih podataka u distribuiranom sistemu, zbog pada u delu komunikacione mreže ili na određenom čvoru u distribuiranom sistemu, uz određene strategije moguće je preusmeriti korisnički zahtev na druge delove distribuiranog sistema i uspešno doći do željenih podataka.

2.3 Efikasnost i fleksibilnost

Efikasnost i fleksibilnost distribuiranog sistema zasniva se na dve činjenice. Prvo, zahvaljujući praticioniranju odnosno ponavljanju podataka u čvorovima distirbuiranog sistema, podaci su fizički blizu onome ko ih stvara i koristi tako da je znatno smanjena potreba za udaljenom komunkacijom. Drugo, pošto svaka lokacija u distribuiranom sistemu uglavnom obrađuje jedan podskup podataka, smanjeno je opterećenje CPU-a kao i ulazno/izlaznih uređaja.

2.4 Veći kapacitet i postupni rast

Čest razlog za uvođenje distribuiranog sistema je nemogućnost jednog računara da primi i obrađuje sve potrebne podatke. U slučaju potrebe za većim kapacitetima, dodavanje čvora distribuiranom sistemu znatno je jednostavnije nego zamena celog centralizovanog sistema većim.

2.5 Lokalna autonomija podataka, upravljanja i kontrole

Okruženje u kome se distribuirani sistemi primenjuju obično je i samo logički i fizički distribuirano. Distribuiranje baza podataka kao i sistema za upravljanje njima, omogućava pojedinim grupama korisnika da lokalno kontrolišu sopstvene podatke, uz mogućnost pristupa podacima na drugim lokacijama kada je to potrebno.

2.6 Problemi u distribuiranom sistemu

Problemi sa kojima se susreću sistemi baza podataka dobijaju na složenosti sa uvođenjem distribuiranog okruženja i ujedno dovode do novih problema izazvanih uglavnom jednim od sledeća tri faktora.

Prvo, distribuirana baza podataka može biti dizajnirana tako da se cela baza ili neki njeni delovi nalaze na većem broju čvorova komunikacione mreže. Potencijalno ponavljanje podataka dovodi do toga da sistem mora da odlučuje o tome odakle će

pribaviti tražene podatke kao i da vodi računa o tome da se efekat promene nekog podatka sprovede nad svim kopijama tog podatka u distribiranom sistemu.

Drugo, ukoliko jedan od čvorova postane iz nekog razloga nedostupan ili se desi pad u komunikacionoj mreži dok je u toku operacija ažuriranja podataka, sistem mora da osigura da će se ažuriranje izvršiti i na podacima trenutno nedostupnim, čim se sistem oporavi od pada.

I konačno, s obzirom da je nemoguće da svaki čvor u distribuiranom sistemu istovremeno dobije informaciju o transakcijama koje se izvršavaju na drugim čvorovima, sistem mora da obezbedi efikasnu sinhronizaciju transakcija, što predstvalja daleko teži zadatak nego u centralizovanom sistemu.

Jedan od problema distribuiranog sistema iskazan je C.A.P. teoremom (Brewers CAP Theorem on distributed systems [19]):

"U distribuiranom sistemu nije moguće istovremeno ispuniti sva tri zahteva:

1. Konzistentnost (engl. *Consistency*)
2. Dostupnost (engl. *Availability*)
3. Otpornost na otkaz (engl. *Partition tolerance*)".

2.7 Organizacija podataka u distribuiranom sistemu

Podaci u distribuiranom sistemu mogu biti particionirani (eng. *partitioned*) ili ponovljeni (eng. *replicated*), a mogu biti istovremeno i particionirani i ponovljeni.

U slučaju particioniranja podataka, baza je razdeljena na više disjunktnih delova od kojih je svaki na drugom čvoru komunikacione mreže. Skup podataka sa logičkog nivoa treba na neki način podeliti, a zatim te delove – fragmente raspodeliti po raznim lokacijama. Logički skup podataka u relacionom sistemu je relacija, a prirodni fragment relacije koji je opet relacija jeste neki njen podskup definisan uslovom projekcije i restrikcije. Fragmentacija mora biti izvedena tako da se spajanjem fragmenata može dobiti polazna relacija. Najčešće primenjivana tehnika za očuvanje informacija pri fragmentaciji podataka je uvođenje sistemskih identifikatora n-torki (nametnuti ključevi) kao primarnih ključeva, koji se pamte uz svaki deo pojedine n-torke i omogućuju njenu rekonstrukciju.[2]

U slučaju ponavljanja podataka, jedan logički objekat može imati više fizičkih reprezentacija (veći broj kopija) na većem broju lokacija. Ukoliko je na svakom od čvorova fizička reprezentacija čitave baze onda je to distribuirani sistem sa potpuno ponovljenim podacima, a inače je distribuirani sistem sa delimično ponovljenim podacima. Distribuirani sistem sa ponovljenim podacima će detaljnije biti razmatran u posebnom poglavlju.

Prilikom izbora načina organizacije podataka u distribuiranom sistemu, teži se smanjenju troškova čuvanja podataka, obrade transakcija kao i komunikacije među čvorovima.

2.8 Upravljanje katalogom

Katalog je sistemska baza podataka koja sadrži podatke o baznim relacijama, pogledima, indeksima, korisnicima itd, a u slučaju distribuiranog sistema, i o načinu i lokacijama na koje su podaci razdeljeni i ponovljeni. Sam katalog u distribuiranom

sistemu može biti centralizovan (samo na jednoj lokaciji), potpuno ponovljen (na svim lokacijama po jedna kopija kataloga), particioniran (na svakoj lokaciji je deo kataloga koji se odnosi na objekte sa te lokacije) ili kombinovan (katalog je particioniran, ali na jednoj lokaciji postoji i centralna kopija kompletnog kataloga).

Svaki od navedenih pristupa ima svojih mana (zavisnost od centralne lokacije, visoka cena prenošenja ažuriranja kataloga ili skup pristup udaljenoj lokaciji) pa pri izboru načina organizacije treba uzeti u obzir karakteristike samog sistema npr. vreme odziva, veličinu kataloga, kapacitet lokacije, pouzdanost itd.

2.9 Distribuirana obrada upita

Distribuirana obrada upita podrazumeva distirbuiranu optimizaciju upita kao i distribuirano izvršavanje upita. Strategije optimizacije upita nad distribuiranom bazom podataka imaju za cilj da minimizuju cenu obrade i vreme za koje će korisnik dobiti odgovor. U troškovima obrade najveću stavku čine troškovi mrežne komunikacije dok troškovi komunikacije sa ulazno/izlaznim uređajima i korišćenja procesora mogu biti i za red veličine niži. Zbog toga je veoma značajno, u zavisnosti od propusnosti mreže i vremena kašnjenja, pravilno odabrati relacije i njihove fragmente koji će biti prenošeni sa jedne lokacije na drugu u cilju obrade upita što predstavlja globalnu optimizaciju. Razlog za prenošenje podataka može biti to što su podaci na lokaciji različitoj od one na kojoj se postavlja upit, ili što u upitu učestvuje veći broj relacija sa različitih lokacija. Izbor strategije za izvršenje operacija na jednoj lokaciji poznat je kao lokalna optimizacija.

2.10 Preneto ažuriranje

Ukoliko je distribuirana baza u celosti ili delimično ponovljena, potreba za konzistentnošću svih kopija podrazumeva da se ažuriranje jednog logičkog objekta mora preneti na sve fizičke kopije tog objekta. Postoji više strategija koje se primenjuju u zavisnosti od toga da li je prioritet konzistentnost podataka ili brzina izvršenja transakcije i o njima će biti više reči u delu o strategijama prenetog ažuriranja u distribuiranom sistemu sa ponovljenim podacima.

2.11 Kontrola konkurentnog izvršavanja i obrada uzajamnog blokiranja

Problem konkurentnog izvršavanja jedan je od najšire izučavanih problema u oblasti distribuiranih sistema baza podataka, a podrazumeva sinhronizaciju pristupa delovima distribuiranog sistema tako da integritet baze ostane očuvan. Pored konzistentnosti baze u smislu u kom se koristi u centralizovanom sistemu, ovde postoji problem održavanja globalne konzistentnosti tj. održavanja svih delova i kopija baze konzistentnim.

Dva glavna pristupa rešavanju ovog problema su pesimistički, gde se sinhronizacija korisničkih zahteva vrši pre nego što izvršenje počne, i optimistički, pri čemu se posle izvršenja ispituje da li je stanje baze konzistentno. Dve tehnike koje se koriste u kombinaciji sa oba prethodno navedena pristupa su zaključavanje koje se bazira na uzajamnom isključivanju pristupa objektima u bazi i tehnika vremenskih oznaka koja podrazumeva uređeno izvršavanje transakcija na svim lokacijama u sistemu.

Problem uzajamnog blokiranja (engl. *deadlock*) u distribuiranom sistemu rešava se metodama prevencije/sprečavanja i otkrivanja/otklanjanja.

3 DISTRIBUIRANI SISTEM SA PONOVLJENIM PODACIMA

Kao što je ranije pomenuto, podaci u distribuiranom sistemu mogu biti particionirani ili ponovljeni ili istovremeno i particionirani i ponovljeni. Ovde će biti detaljnije razmotren distirbuirani sistem sa ponovljenim podacima.

Razlozi za ponavljanje podataka su:

1. Dostupnost sistema – Ponavljanjem podataka otklanja se mogućnost pada čitavog sistema zbog pada jedne od lokacija jer su podaci dostupni sa drugih lokacija.
2. Performanse - Kao što je već diskutovano, u distribuiranom sistemu jednu od najvećih stavki u troškovima obrade predstavljaju troškovi mrežne komunikacije. Ponavljanje podataka omogućava da podatke smestimo na lokaciju odakle im se i pristupa čime je pristup lokalizovan, u velikoj meri eliminisana mrežna komunikacija, a time i vreme pristupa smanjeno.
3. Veći postupni rast - U slučaju rasta sistema u smislu povećanja broja lokacija, ponavljanje podataka omogućava jednostavno uključivanje novog čvora u distribuirani sistem.
4. Zahtevi programa – Ponavljanje podataka može biti diktirano potrebama programa, koji može imati potrebu da održava kopije podataka kao deo specifikacije problema čije se rešenje implementira.

Iako su prednosti očigledne, ponavljanje podataka nameće problem održavanja kopija sinhronizovanim tj. problem distribuiranog upravljanja transkacijama.

Na način funkcionisanja distribuiranog sistema baza podataka sa ponovljenim podacima utiču sledeći faktori: [1]

- Dizajn baze podataka - Kao što je već pomenuto, baza može biti potpuno ili delimično ponovljena. U slučaju delimično ponovljenih podataka, broj fizičkih kopija objekata u bazi za svaki logički objekat može da varira, a u nekim slučajevima objekat ne mora uopšte biti ponovljen. S tim u vezi, transakcije koje se izvršavaju na neponovljenim podacima su lokalne transakcije i njihovo izvršenje neće biti predmet ovog rada. Sa druge strane, transakcije koje se tiču ponovljenih podataka moraju biti izvršene na različitim lokacijama i nazivaju se globalnim transakcijama.
- Konzistentnost baze – Kada globalne transakcije ažuriraju kopije podataka na različitim lokacijama, vrednosti tih kopija mogu biti različite u različitim vremenskim trenucima. Distribuirana baza je u globalno konzistentnom stanju ako sve kopije svakog objekta u bazi imaju iste vrednosti. Ono što razlikuje različite kriterijume globalne konzistentnosti jeste stepen sinhronizovanosti kopija (jaki i slabi kriterijumi globalne konzistentnosti).
- Gde se odvija ažuriranje – Strategije ažuriranja mogu biti centralizovane, ukoliko se ažuriranje prvo sprovodi na master kopiji, ili distribuirane, ukoliko se ažuriranje sprovodi na bilo kojoj kopiji.
- Preneto ažuriranje – Kada se ažuriranje obavi na jednoj kopiji (master ili bilo kojoj drugoj) postavlja se pitanje kako preneti ažuriranje na ostale kopije. Postoje dve alternative, a to su vredno (engl. *eager*) i lenjo (engl. *lazy*) preneto ažuriranje. Tehnika vrednog ažuriranja izvodi sva ažuriranja u okviru globalne transakcije,

čime su sve kopije automatski ažurirane. Tehnika lenjog ažuriranja, sa druge strane, obavlja ažuriranja ostalih kopija posle završetka inicijalne transakcije.

- Stepen nevidljivosti ponavaljanja podataka – Neki protokoli omogućavaju samo ograničenu nevidljivosti ponavaljanja podataka tako što zahtevaju da korisnik i njegov program znaju za master lokaciju (sa master kopijom) i da nad njom izvršavaju svoje transakcije. Drugi protokoli omogućavaju potpunu nevidljivosti ponavaljanja podataka tako što uključuju upravljač transakcijama na svakoj lokaciji, čime je omogućeno da korisnik transakcije izvršava na bilo kojoj lokaciji u sistemu.

3.1 Globalna i transakciona konzistentnost

Postoje dva pitanja vezana za konzistentnost distribuirane baze sa ponovljenim podacima. Jedno je već pomenuta globalna konzistentnost, koja se odnosi na problem jednakosti fizičkih kopija objekta sa jednim logičkim objektom. Drugo pitanje se tiče serijskog izvršenja transakcija koje treba ponovo razmotriti u kontekstu distribuiranog sistema. I dodatno, treba razmotriti povezanost i međusobni uticaj ova dva pitanja.

3.1.1 Transakciona konzistentnost

Pod transakcijom u distribuiranom sistemu podrazumeva se, kao i u centralizovanom sistemu, vremenski uređen niz radnji koji prevodi jedno konzistentno stanje baze u drugo. Ipak, pojam transakcije u distribuiranom sistemu je kompleksniji, s obzirom da transakcija može da izvršava radnje svog programa na raznim lokacijama; zato transakciju može da izvršava veći broj procesa na većem broju lokacija. Prilikom distribuirane obrade transakcija, transakcija koja čita vrednost objekta čita, u zavisnosti od organizacije sistema, kopiju objekta sa lokacije na kojoj je transakcija i pokrenuta ili sa neke druge lokacije (master lokacija ili primarna kopija), dok transakcija koja ažurira objekat, sprovodi ažuriranje na svim kopijama objekta. [2]

U distiribuiranom sistemu razlikujemo lokalno i globalno izvršenje skupa transakcija.

Neka je dat skup transakcija $T = \{T_i\}^n_{i=1}$ nad bazom podataka distribuiranom na k lokacija.

Lokalno izvrešenje skupa transakcija T na lokaciji l je niz I_l trojki oblika (T_j, pročitaj, x_l), odnosno (T_j, ažuriraj, x_l), gde je T_j iz T, x_l je primerak objekta x na lokaciji l, a radnja pročitaj x odnosno ažuriraj x, radnja transakcije T_j. Poredak trojki u ovom nizu odgovara poretku radnji pojedine transakcije.

Distribuirano izvršenje skupa T je niz lokalnih izvršenja $I= \{I_i\}^n_{i=1}$, takav da važi:

- (čitanje samo jednog primerka): ako je (T_i, pročitaj, x_j) iz I_j, tada (T_i, pročitaj, x_l) ne pripada I_l za $l \neq j$, gde su x_j, x_l primerci objekta x na lokacijama j, l redom;
- (ažuriranje svih primeraka): ako je (T_i, ažuriraj, x_j) iz I_j, tada (T_i, ažuriraj, x_l) iz I_l za sve lokacije l na kojima postoji primerak x_l objekta x;
- Svakoj radnji svake transakcije T_j odgovara bar jedna trojka sa prvom komponenetom T_j.

Kao kod centralizovanih sistema, i kod distribuiranih sistema mogu se definisati serijska distribuirana izvršenja, ekvivalentna distribuirana i linearizovana (ekvivalentna serijskim) distribuirana izvršenja skupa transakcija.

Serijsko distribuirano izvršenje skupa transakcija $T = \{T_i\}^n_{i=1}$ je distribuirano izvršenje I za koje se na skupu T može definisati uređenje $<$ za koje važi: ako je $T_i < T_j$ onda sve radnje transakcije T_i prethode svim radnjama transakcije T_j u svakom lokalnom izvršenju I_l u kojem se pojavljuju radnje obeju transakcija.

Dve radnje skupa transakcija T su konfliktne u distribuiranom izvršenju ako su konfliktne (u centralizovanom smislu) u nekom (bilo kome) lokalnom izvršenju.

Dva distribuirana izvršenja su ekvivalentna ako su, za svaku lokaciju, njihova lokalna izvršenja ekvivalentna (u centralizovanom smislu).

Distribuirano izvršenje je linearizovano ako je ekvivalentno nekom serijskom izvršenju.

Jedna metoda za postizanje linearizovanog distribuiranog izvršenja je, kao i kod centralizovanog linearizovanog izvršenja, dvofaznost transakcija, koja se u distribuiranom slučaju proširuje sledećim zahtevima:

- pri čitanju logičkog objekta x, dovoljno je da transakcija zaključa deljivim katancem jedan primerak objekta x (onaj koji stvarno čita);
- pri ažuriranju logičkog objekta x, transakcija mora da postavi ekskluzivne katance na sve primerke objekta x, na svim lokacijama u distribuiranoj bazi;
- ako transakcija ne može da dobije traženi katanac, ona staje u red za čekanje za onaj primerak objekta nad kojim treba da izvrši radnju.

Glavni nedostatak prethodnog postupka dvofaznog zaključavanja je što transakcija, pri ažuriranju jednog logičkog objekta, mora dobiti ekskluzivni katanac na svim primercima tog objekta, na raznim lokacijama, a o dodeljivanju i oslobađanju katanaca odlučuju lokalni upravljači zaključavanja. Zato taj postupak zahteva puno komunikacije (slanja i primanja poruka), i može se pojednostaviti na nekoliko načina (od kojih svaki ima svoje prednosti i nedostatke).

Ukoliko je ispunjen uslov da je globalno izvršenje serijsko ili linearizovano, baza će biti i globalno konzistentna, dok obrnuto ne mora da važi.

Slabiji kriterijum, koji se pojavljuje u sve većem broju komercijalnih rešenja, je izolacija poslednjeg stanja (engl. *snapshot isolation, SI*). Ovaj kriterijum omogućava transakciji T da čita nepotvrđene promene drugih transakcija, kao i drugim transakcijama da pristupe podacima koje transakcija T upravo čita. Kao posledica, operacija čitanja nikada nije blokirana od strane operacije pisanja čime se poboljšavaju performanse sistema, ali zato izvršenje nije linearizovano.

3.1.2 Globalna konzistentnost

Kritetijum za globalnu konzistentnost može biti jak ili slab i svaki je pogodan za različite klase programa sa različitim potrebama.

Jak kriterijum globalne konzistentnosti podrazumeva da sve kopije logičkog objekta x imaju istu vrednost po završetku transakcije. Ovo se uglavnom obezbeđuje primenom dvofaznog zaključavanja.

Slaba globalna konzistentnost podrazumeva da vrednosti svih kopija objekta x u nekom trenutku postanu jednake. Teško je precizno definisati ovaj koncept, i najpreciznija definicija bila bi sledeća:

- U svakom trenutku, za svaku kopiju, postoji istorija koja je ekvivalentna istoriji svake druge kopije objekta x. Ova istorija se naziva potvrđena istorija kopije.
- Potvrđena istorija svake kopije monotono raste.
- Sve neponištene operacije u potvrđenoj istoriji zadovoljavaju svoje preduslove.
- Za svaku operaciju A, ili A ili njeno poništenje će u nekom trenutku biti uključeno u potvrđenu istoriju.

Ova definicija je prilično stroga i u praksi se uglavnom koriste neki slabiji uslovi. Čak je ostavljeno da korisnik postavi sopstvene kriterijume za "svežinu" kopije, koji odgovaraju specifičnom programu. Neki od tih kriterijuma su:

- Kriterijum vremenskog ograničenja: korisnik definiše vremenski okvir u kojem je dopustiva globalna nekonzistentnost baze.
- Kriterijum ograničenja vrednosti: korisnik definiše opseg u kome vrednost kopija može da se razlikuje.

Važan parametar za analizu protokola globalne konzistentnosti je "stepen svežine" kopije x_i u trenutku t i definiše se kao broj ažuriranja sprovedenih nad kopijom x_i do trenutka t u odnosu na ukupan broj ažuriranja.

3.2 Strategije ažuriranja podataka

3.2.1 Strategija vrednog prenošenja ažuriranja

Strategija vrednog prenošenja ažuriranja (engl. ***e****ager update propagation*) podrazumeva ažuriranje svih kopija u okviru globalne transakcije. Samim tim, kada se transakcija izvrši, sve kopije imaju istu vrednost. Ova strategija prenetog ažuriranja uglavnom koristi protokol dvofaznog potvrđivanja (engl. *Two Phase Commit - 2PC protocol*), ali su moguća i druga rešenja. Strategija vrednog prenošenja ažuriranja može biti sinhrona, kada se sve kopije ažuriraju istovremeno, i odložena (engl. *deferred*), kod koje se ažuriranje kopije sprovodi onda kada je to potrebno, dok se ažuriranje ostalih kopija odlaže za kraj transakcije. Odloženo izvršenje se može implementirati uključivanjem ažuriranja u "Prepare-to-Commit" poruku na početku izvršenja dvofaznog potvrđivanja.

Ova strategija uzrokuje jaku globalnu konzistentnost. S obzirom na to da po završetku transakcije sve kopije imaju istu vrednost, naredna Read(x) operacija može da se izvrši na bilo kojoj kopiji, dok se, kao što je već rečeno, Write(x) operacija mora izvršiti nad svim kopijama objekta x. Zato se protokoli koji podržavaju ovaj vid prenetog ažuriranja nazivaju ROWA (Read-one/write-all) protokoli.

Za postizanje globalne konzistentnosti koristi se kriterijum serijskog izvršenja na jednoj kopiji (engl. *one-copy serializability, 1SR*). Ovaj kriterijum podrazumeva da efekti transakcija na ponovljenim podacima budu isti kao da se transakcije izvršavaju serijski na jednoj kopiji podataka. Drugim rečima, izvršenje treba da bude ekvivalentno nekom serijskom izvršenju na neponovljenim podacima.

Prednosti ove strategije su višestruke. Prvo, pošto za postizanje globalne konzistentnosti koriste 1SR kriterijum, postignuta je i transakciona konzistentnost. Drugo, transakcija može da čita lokalnu kopiju objekta x i da bude sigurna da je to ažurna verzija, čime je smanjena potreba za udaljenom komunikacijom. Konačno, promene na kopijama se obavljaju automatski pa se i oporavak od pada može realizovati na mnogo jednostavniji i brži način.

Najveća mana strategije vrednog prenošenja ažuriranja je u tome što je potrebno ažurirati sve kopije da bi se transakcija završila. Ovo za posledicu ima pad performansi čitavog sistema, jer brzina izvršenja ažuriranja zavisi od najsporijeg čvora u sistemu (u smislu brzine izvršenja operacije ažuriranja). Takođe, ukoliko je neka od kopija trenutno nedostupna, transakcija ne može da se završi.

3.2.2 Strategija lenjog prenošenja ažuriranja

Za razliku od strategije vrednog prenošenja ažuriranja, pri strategiji lenjog prenošenja ažuriranja (engl. *Lazy update propagation*) transakcija se završava čim se izvrši na lokaciji gde je i pokrenuta. Ažuriranje ostalih kopija se obavlja asinhrono, tako što se šalje transakcija osvežavanja (engl. *refresh transaction*) ostalim kopijama, nekada i pošto se inicijalna transakcija već izvršila na lokaciji gde je pokrenuta. Transakcija osvežavanja sadrži niz operacija koje odgovaraju originalnoj transakciji.

Ova strategija koristi se u situacijama kada globalna konzistentnost nije prioritet i kada je moguće tolerisati nekonzistentnost kopija na račun boljih performansi sitema.

Glavna prednost ove strategije je kraće vreme operacije ažuriranja, pošto je transakcija završena čim je završena na jednoj lokaciji. Mana je to što kopije nisu globalno konzistentne i što neke od njih mogu biti neažurne, tako da se ne može garantovati da će lokalna Read(x) operacija vratiti ažurnu vrednost. Dalje, može se desiti da $Read_i(x)$ operacija transakcije T_i ne vidi efekte $Write_j(x)$ prethodne transakcije T_j ($j < i$) jer kopija nije ažurna. Ova pojava naziva se inverznost transakcija i može se izbeći primenom 1SR i SI strategija, ali je to skupo rešenje.

3.2.3 Centralizovane strategije

Kod centralizovanog prenetog ažuriranja, ažuriranje se prvo izvršava na master kopiji, a zatim prenosi na ostale kopije podanike (engl. *slaves*). Lokacija koja sadrži master kopiju naziva se master lokacija, dok se ostale lokacije nazivaju lokacije podanici. U nekim slučajevima postoji samo jedna master kopija za sve ponovljene objekte i takve strategije se nazivaju centralizovane strategije sa jednom master kopijom. U slučaju da svaki ponovljeni objekat (ili češće, grupa ponovljenih objekata) ima sopstvenu master kopiju (za objekat x, master kopija može biti x_i na lokaciji S_i, a za objekat y, master kopija je y_j na lokaciji S_j) strategija se naziva strategijom primarne kopije (engl. *primary copy*).

Prednosti centralizovanih strategija prenetog ažuriranja su u tome što se ažuriranje dešava samo na master kopiji, pa ne postoji potreba za distribuiranom obradom transakcija i to što je sigurno jedna lokacija, ona sa master kopijom, uvek ažurna.

Ono što je loše kod ovih strategija je ono što je inače mana svih centralizovanih algoritama, a to je problem preopterećenja ili pada centralne lokacije tj. master kopije.

Strategija primarne kopije je jedan od načina da se ovaj problem redukuje, ali se zato povećava problem konzistentnosti jer postoji potreba za sinhronizacijom među kopijama, što spada u probleme tehnike lenjog ažuriranja opisane ranije.

3.2.4 Distribuirane strategije

Distribuirane strategije primenjuju operaciju ažuriranja na kopiju objekta na lokaciji gde je transakcija i pokrenuta, a zatim se operacija prenosi i na druge lokacije na kojima postoje kopije objekta. Ovo podrazumeva da različite transakcije mogu da ažuriraju kopije istog objekta x na različitim lokacijama što dovodi do problema u održavanju konzistentnosti. Ukoliko se ove tehnike primenjuju zajedno sa strategijom vrednog prenošenja ažuriranja, održavanje konzistentnosti se može relativno lako rešiti. Ali, ukoliko se koriste strategije lenjog prenošenja ažuriranja može doći do toga da se transakcije izvode u različitom redosledu na različitim lokacijama čime globalno izvršenje nije serijsko. Veći problem predstavlja to što po završetku usklađivanja (završetku svih transakcija na svim lokacijama) kopije neće imati iste vrednosti objekata. Rešenje ovog problema leži u poništavanju i ponovnom pokretanju nekih transakcija sa ciljem postizanja serijskog izvršenja na svakoj lokaciji. Ovo rešenje dodatno komplikuje sistem, a i zavisi od toga da li aplikacija dozvoljava poništavanje i ponono pokretanje transakcija.

3.3 Protokoli

3.3.1 Centralizovani protokoli sa strategijom vrednog prenošenja ažuriranja

Kod centralizovanih protokola sa strategijom vrednog prenošenja ažuriranja, postoji master lokacija koja kontroliše sve operacije na objektu, a ažuriranja se prenose na sve kopije objekta u okviru jedne transakcije korišćenjem dvofaznog protokola potvrđivanja. Po završetku transakcije, sve kopije imaju istu vrednost i globalno izvršenje je serijsko. Postoje dva parametra koja razlkuju specifične implementacije ovih protokola: gde se izvršavaju ažuriranja i stepen nevidljivosti ponavljanja podataka. Prvi parameter određuje da li postoji jedna master kopija za sve objekte koji se ponavljaju (engl. *single master*) ili različite lokacije sadrže master kopiju objekta, ili češće grupe objekata (engl.*primary copy*) koji se ponavljaju. Drugi parametar određuje da li korisnički program zna za postojanje master kopije (samo na njoj izvršava transakcije) ili može da se osloni na lokalni upravljač transakcijama i izvršava transakcije sa bilo koje lokacije u sistemu.

3.3.1.1 Jedna master lokacija sa ograničenom nevidljivošću ponavljanja podataka

Najjednostavniji slučaj je imati jednu master kopiju za sve ponovljene objekte pri čemu korisnik zna za njeno postojanje. U tom slučaju se globalne transakcije koje sadrže bar jednu operaciju pisanja prosleđuju direktno master kopiji, tj. upravljaču transakcijama na master lokaciji. Na master lokaciji, svaka operacija čitanja objekta x se izvršava tako što se x_M (M predstavlja oznaku master kopije) zaključava, čita se objekat x_M i rezultat se vraća korisniku. Slično se izvršava i operacija pisanja na master kopiji, a zatim upravljač

transakcijama na master lokaciji prosleđuje operaciju pisanja ostalim lokacijama u sistemu, istovremeno ili na neki drugi način. Kod prosleđivanja operacije pisanja ostalim lokacijama, potrebno je voditi računa o tome da redosled izvršavanja konfliktnih operacija pisanja bude isti kao i na master kopiji, što se može postići, na primer, primenom vremenskih oznaka. Transakcije koje sadrže samo operacije čitanja, mogu se pokretati i na lokacijama koje nisu master lokacija tako što se operacije čitanja prosleđuju master kopiji na kojoj se objekat koji se čita zaključava, pročita se vrednost i vrati se korisniku ili master kopija samo pošalje poruku da je objekat zaključan, a čita se lokalna vrednost objekta.

3.3.1.2 Jedna master lokacija sa punom nevidljivošću ponavljanja podataka

Protokoli sa jednom master kopijom i strategijom vrednog prenošenja ažuriranja zahtevaju od korisničke aplikacije da zna za ovakvu organizaciju i stvara veliko opterećenje na master lokaciji u smislu broja zahteva, izvršenja operacija i komunikacije. Puna nevidljivost ponavljanja podataka može se postići tako što se transakcije ne prosleđuju direktno master lokaciji, već njihovu obradu preuzima lokalni upravljač transakcijama koji dalje koordinira njenim izvršenjem. Na taj način se korisnički program oslanja na lokalni upravljač transakcijama i ne mora da zna za postojanje master kopije. Ovakvo rešenje, iako lako za implementaciju, ne rešava problem preopterećenosti master kopije. Alternativno rešenje bi izgledalo ovako:

- Upravljač transakcijama koji koordiniše izvršenje transakcije šalje svaku operaciju, redom, master lokaciji.
- Ukoliko je operacija čitanje, postavlja se deljivi katanac na master kopiju objekta koji se čita i obaveštava se koordinator o tome. Koordinator zatim prosleđuje operaciju čitanja bilo kojoj lokaciji na kojoj postoji kopija objekta x, gde operaciju obrađuje lokalni sistem.
- Ukoliko je operacija pisanje, master lokacija se ponaša na sledeći način:
    - Postavlja isključivi katanac na master kopiju objekta x
    - Izvršava operaciju pisanja na master kopiji objekta x
    - Obaveštava koordinatora transakcijom da je postavljen isključivi katanac na objekat x

Koordinator dalje šalje operaciju pisanja ostalim lokacijama koje sadrže kopiju objekta x, koja se tamo lokalno obrađuje.

Najbitnije postignuto ovim rešenjem je rasterećenje master kopije, jer je obrada operacija čitanja kao i koordinacija operacija pisanja prebačena na lokalne upravljače transakcijama.

3.3.1.3 Primarna kopija sa punom nevidljivošću ponavaljanja podataka

Kao što je ranije rečeno, u ovoj tehnici svaki logički objekat ima jedan svoj primarni fizički objekat, i moguće je, veći broj ostalih kopija. Razni logički objekti mogu imati primarne objekte na raznim lokacijama. Upravljač zaključavanja lokacije na kojoj je primarni objekat logičkog objekta x, obrađuje zahteve za zaključavanjem objekta x (i na svim drugim lokacijama). I u ovom slučaju, svi primerci jednog objekta ponašaju se, u svrhe zaključavanja, kao jedinstveni objekat, ali ova tehnika ne pokazuje nedostatke

centralizovanog upravljača zaključavanja. Nedostatak metode dvofaznosti transakcija u obezbeđivanju linearizovanosti konkurentnog distirbuiranog izvršenja, kao i u slučaju centralizovanih sistema, jeste to što može doći do uzajamnog blokiranja transakcija. Ukoliko su u odnos uzajamnog blokiranja uključeni objekti na više lokacija dolazi do tzv. globalnog uzajamnog blokiranja koje je nemoguće detektovati od strane lokalnih upravljača zaključavanja jer u lokanim grafovima čekanja nema ciklusa.

Jedan način za detekciju uzajamnog blokiranja je centralizovanje jedne lokacije i periodično prebacivanje lokalnih grafova čekanja sa svih drugih lokacija na centralnu lokaciju. Međutim, mana ovog rešenja je mogućnost pada centralne lokacije. Postoji i bolje, ali kompleksnije rešenje koje podrazumeva dodavanje novog čvora u lokalni graf čekanja pojedinačne lokacije I, koji bi predstavljao svaku transakciju koja čeka na kompletiranje neke druge transakcije sa lokacije I, odnosno svaku transakciju na čije kompletiranje čeka neka transakcija lokacije I. Ako ni u jednom tako proširenom grafu čekanja lokacije ne postoji ciklus, globalnog uzajamnog blokiranja nema u celom sistemu; a ako takav ciklus postoji, onda je to indikator mogućeg globalnog uzajamnog blokiranja u celom sistemu.

3.3.2 Distribuirani protokoli sa strategijom vrednog prenošenja ažuriranja

Kod ovih protokola transakcija može biti pokrenuta sa bilo koje lokacije u sistemu, i prvo se izvršava na lokalnoj kopiji, a zatim se prosleđuje na ostale lokacije. Ukoliko je transakcija pokrenuta na lokaciji na kojoj nema kopije traženog objekta, transakcija se prosleđuje nekoj od lokacija na kojoj kopija postoji, i ona dalje koordinira izvršenjem transakcije. Kao i kod ostalih vrednih tehnika prenetog ažuriranja, sve ovo se dešava u okviru jedne transakcije, po čijem završetku sve kopije objekta imaju istu vrednost.

Ono što predstavlja teškoću u primeni ovakvog protokola je postizanje linearizovanog distribuiranog izvršenja. Ovo se razrešava tehnikama konkurentne kontrole primenjene na svakoj lokaciji.

3.3.3 Centralizovani protokoli sa strategijom lenjog prenošenja ažuriranja

Protokoli ovog tipa slični su centralizovanim protokolima sa strategijom vrednog prenošenja ažuriranja po tome što se transakcije koje sadrže operaciju pisanja prvo primenjuju na master kopiju, a zatim prosleđuju ostalim kopijama. Razlika je u tome što se kod centralizovanih protokola sa strategijom vrednog prenošenja ažuriranja, ažuriranje svih kopija obavalja u okviru transakcije u kojoj se ažurira i master kopija, dok se kod centralizovanih protokola sa strategijom lenjog prenošenja ažuriranja, ostale kopije ažuriraju po završetku transakcije na master kopiji, uglavnom u vidu posebnih transakcija osvežavanja.

3.3.3.1 Jedna master lokacija sa ograničenom nevidljivošću ponavaljanja podataka

U ovom slučaju, transakicije koje sadrže bar jednu operaciju pisanja se pokreću i izvršavaju na master kopiji, a kada se transakcija završi (potvrdi) transakcija osvežavanja se šalje ostalim kopijama.

Ako se operacija čitanja zada na kopiji koja nije master, čita se lokalna kopija i rezultat se vraća korisniku. Ovde treba primetiti da ova vrednost ne mora da bude ažurna, ukoliko je na master kopiji izvršeno ažuriranje, a transakcija osvežavanja još nije stigla do date kopije. Ukoliko se operacija pisanja zada na kopiji koja nije master, operacija se poništava.

U slučaju primarne kopije sa ograničenom nevidljivošću ponavljanja podataka situacija je slična, samo se umesto na master lokaciji, transakcije zadaju na primarnoj kopiji objekta x.

Kod ovog protokola postoji problem u redosledu izvršavanja transakcija osvežavanja na ne-master kopijama. U slučaju master kopije ovaj problem se može razrešiti tehnikom vremenskih oznaka, dok se u slučaju primarne kopije situacija komplikuje pa postoji više pristupa u rešavanju problema.

Jedan od načina je da se na svaku vremensku oznaku doda i identifikator master kopije sa koje transakcija potiče. Problem je sa onim transakcijama koje ne ispoštuju redosled jer je ne mogu biti poništene, s obzirom da su već izvršene na primarnoj kopiji. Rešenje ovog problema je ili u poništavanju čitave transakcije ili u primeni analize replikacionog grafa. Čvorovi replikacionog grafa sastoje se od transakcija (T) i lokacija (S) pri čemu postoji grana $<T_i , S_j>$ akko T_i treba da izvrši operaciju pisanja na kopiji objekta koja se nalazi na lokaciji S_j. Za svaku zadatu operaciju (op_k) dodaje se odgovarajući čvor T_k i odgovarajuće grane i proverava se da li ima ciklusa. Ukoliko nema ciklusa izvršenje se nastavlja. Ukoliko se detektuje ciklus koji uključuje transakciju koja je već izvršena na primarnoj kopiji, ali čije transakcije osveženja još nisu izvršene na svim kopijama, transakcija T_k se poništava (da bi se kasnije ponovo pokrenula). U suprotnom, T_k sačeka svoj red na izvršenje. Kada se transakcija završi na svim lokacijama, odgovarajući čvorovi i grane se uklanjaju iz grafa. Dokazano je da ovaj protokol obezbeđuje serijsko izvršenje.

3.3.3.2 Master ili primarna kopija sa potpunom nevidljivošću ponavaljanja podataka

Sada ćemo se okrenuti alternativama koje omogućuju potpunu nevidljivost ponavljanja podataka tako što se operacije i čitanja i pisanja mogu zadavati na svim lokacijama i onda prosleđivati ili master kopiji ili odgovarajućoj primarnoj kopiji. Kod ovakvog pristupa postoje dva problema. Prvi je održavanje serijskog izvršavanja, a drugi problem je što može da se desi da transakcija ne vidi rezultat sopstvenog ažuriranja.

Zbog ovih problema ne postoji mnogo mogućnosti za potpunu nevidljivost ponavljanja podataka u lenjm algoritmima. Među njima postoji rešenje Bernstaina [?] koje se odnosi na slučaj sistema sa jednom master kopijom i čije se metode za proveru ispravnosti redosleda izvršenja transakcija odvijaju na master lokaciji, a slične su optimističkom načinu kontrole konkurentnosti. Osnovna ideja je u sledećem: Neka je T transakcija koja vrši pisanje objekta x. U trenutku potvrđivanja transakcije T, master lokacija generiše vremensku oznaku za transakciju T i to postaje i vremenska oznaka master kopije objekta x (x_m) tj . vrednost oznake transakcije koja ga je poslednja ažurirala (last_modified (x_m)). Ova oznaka se dodaje i transakcijama osvežavanja. Kada ne-master kopija x_i primi transakciju osvežavanja, vrednost poslednjeg pristupa kopiji x_i postaje vremenska oznaka transakcije osvežavanja (last_modified(x_i) $\leftarrow$ last_modified(x_m)).

Generisanje vremenske oznake transakcije na master kopiji se odvija po sledećem pravilu: Oznaka transakcije T mora da bude veća od bilo koje već generisane oznake, i mora biti manja od oznake poslednjeg ažuriranja (last_modified) objekata kojima je pristupala. Ukoliko takva oznaka ne može da se generiše, transakcija T se prekida.

Ovaj test osigurava da će operacije čitanja uvek čitati ažurnu vrednost.

Iako ovaj algoritam rešava prvi gore navedeni problem, ne rešava automatski i drugi. Drugi problem se rešava održavanjem liste svih ažuriranja koje je izvršila transakcija i uzimanjem u obzir njenih vrednosti kada se izvršava operacija čitanja. Doduše, kako jedino master kopija ima uvid u liste ažuriranja, one se moraju i održavati na master kopiji čime i sve operacije čitanja moraju da se zadaju samo na master kopiji.

3.3.4 Distribuirani protokoli sa strategijom lenjog prenošenja ažuriranja

Distribuirani protokoli sa strategijom lenjog prenošenja ažuriranja su najkompleksniji jer dopuštaju operacije ažuriranja na bilo kojoj lokaciji u sistemu, a da se zatim strategijom lenjog ažuriranja prosleđuju ostalim lokacijama.

Na lokaciji gde se transakcija zadaje sve funkcioniše jednostavno; i čitanje i upis se izvršavaju lokalno na lokalnoj kopiji. Po potvrđivanju transakcije, ostalim kopijama se šalje transakcija osvežavanja.

Problemi nastaju pri obradi ažuriranja na drugim lokacijama u odnosu na onu gde je transakcija zadata. Kada transakcija osvežavanja stigne na lokaciju, mora biti vremenski raspoređena, čime upravlja lokalni mehanizam za kontrolu konkurentnog izvršavanja. Problem je što više transakcija može da ažurira različite kopije istog objekta konkurentno na različitim lokacijama, pri čemu nastaju konflikti. Redosled izvršenja transakcija osvežavanja se određuje na osnovu rezultata arbitraže posle čega se ažuriraju kopije na svim lokacijama. Može se dizajnirati algoritam arbitraže opšte namene baziran na heuristikama. Na primer, ažuriranja se mogu izvršavati u poretku određenom vremenskim oznakama (npr. najkasnije vremenske oznake pobeđuju) ili se može dati prednost ažuriranjima sa nekih lokacija.

Ovo su *ad-hoc* rešenja, a arbitraža stvarno zavisi najviše od semantike aplikacije. Jednostavan pristup baziran na vremenskim oznakama, koji spaja broj lokacije sa lokalnim vremenom, dovodi do arbitraže između transakcija koja nema stvarne veze sa logikom aplikacije. Takođe, poredak po vremenskim oznakama funkcioniše samo uz pretpostavku da su lokalni satovi sinhronizaovani, što je teško postići u velikim distribuiranim sistemima. Kakva god strategija arbitraže da se upotrebi, neki podaci su izgubljeni.

4 ALATI ZA PONAVLJANJE PODATAKA

4.1 IBM DB2 Data Replication V8

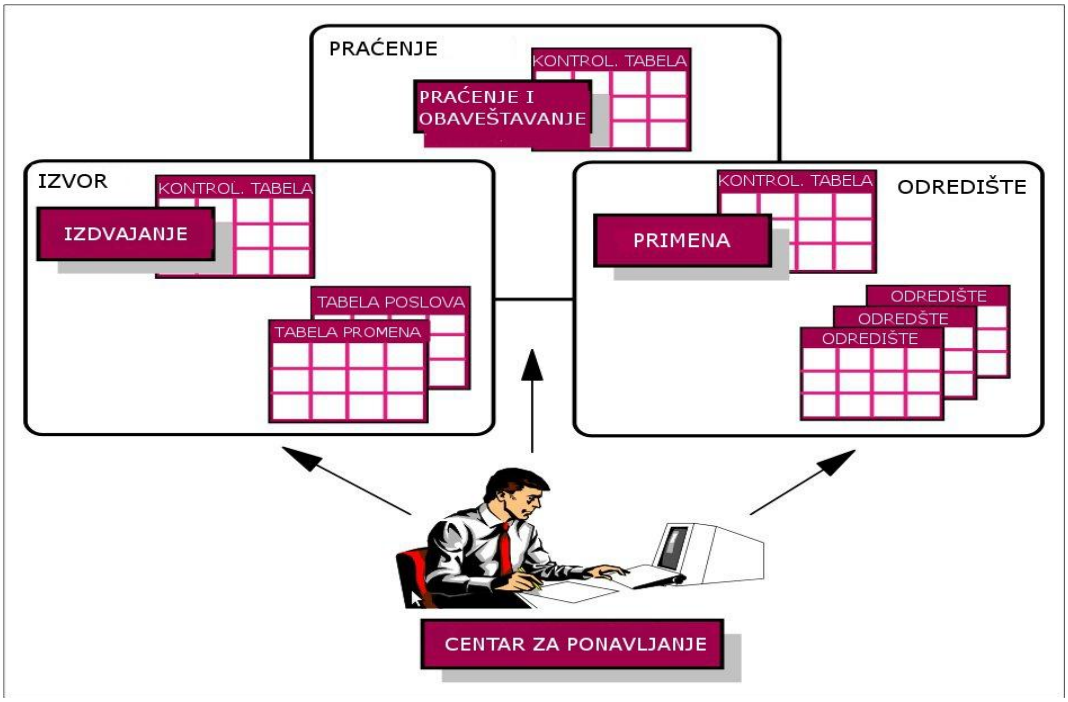
IBM rešenje za distribuirani sistem sa ponovljenim podacima ima 4 komponente:[3]

1. Administracija (engl. *Amdministration*)
2. Izdvajanje (engl. *Capture)*
3. Primena (engl. *Apply*)
4. Praćenje i obaveštavanje (engl. *Alert Monitor)*

Ove četiri komponente komuniciraju preko relacionih tabela, nazvanih kontrolne tabele. Kontrolne tabele se prave i popunjavaju korišćenjem centra za ponavljanje podataka (Replication Center). Komponente za izdvajanje, primenu, praćenje i obaveštavanje čitaju i ažuriraju informacije u kontrolnim tabelama.

4.1.1 Administracija

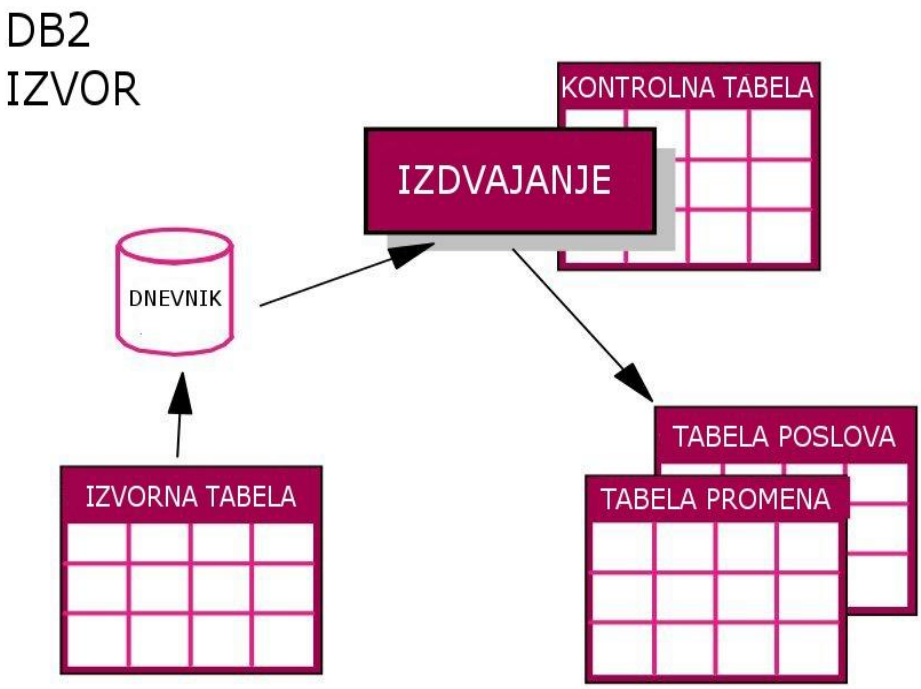
Centar za ponavljanje podataka je grafički korisnički interfejs koji se koristi za definisanje izvora podataka i povezivanja sa odredištima. Takođe se koristi za upravljanje i nadgledanje procesa za izdvajanje i primenu na lokalnim i udaljenim sistemima. Centar za ponavljanje radi na Windows i Unix/Linux sistemima i mora imati vezu ka izvoru kao i ka odredišnim serverima. DB2 V8 Administration Client for Windows and UNIX uključuje i ovu komponentu.



Slika1 Korišćenje centra za ponavljanje podataka

4.1.2 Izdvajanje

Promene na db2 izvornim tabelama izdvaja proces koji je pokrenut na izvornom serveru. DB2 izvorni server može biti DB2 za z/OS i OS/390 verzija 6,7 i 8, DB2 za iSeries na OS/400 V5R2, ili DB2 za Windows i UNIX Version 8. Podaci mogu da se filtriraju po kolonama tokom ovog procesa. Izdvojene promene se čuvaju u tabelama lokalnim za izvornu tabelu i automatski se brišu kada se iskoriste.



Slika 2 Izdvajanje

Kada se dese promene na izvornim tabelama, db2 zapisuje podatak o tome u dnevnik. Ovi zapisi se koriste za oporavak baze i za ponavljanje podataka. Program za izdvajanje promena koristi interfejs baze da pristupi zapisima u dnevniku:

DB2 z/OS and OS/390 IFI 306

DB2 Windows and UNIX asynchronous log read API db2ReadLog

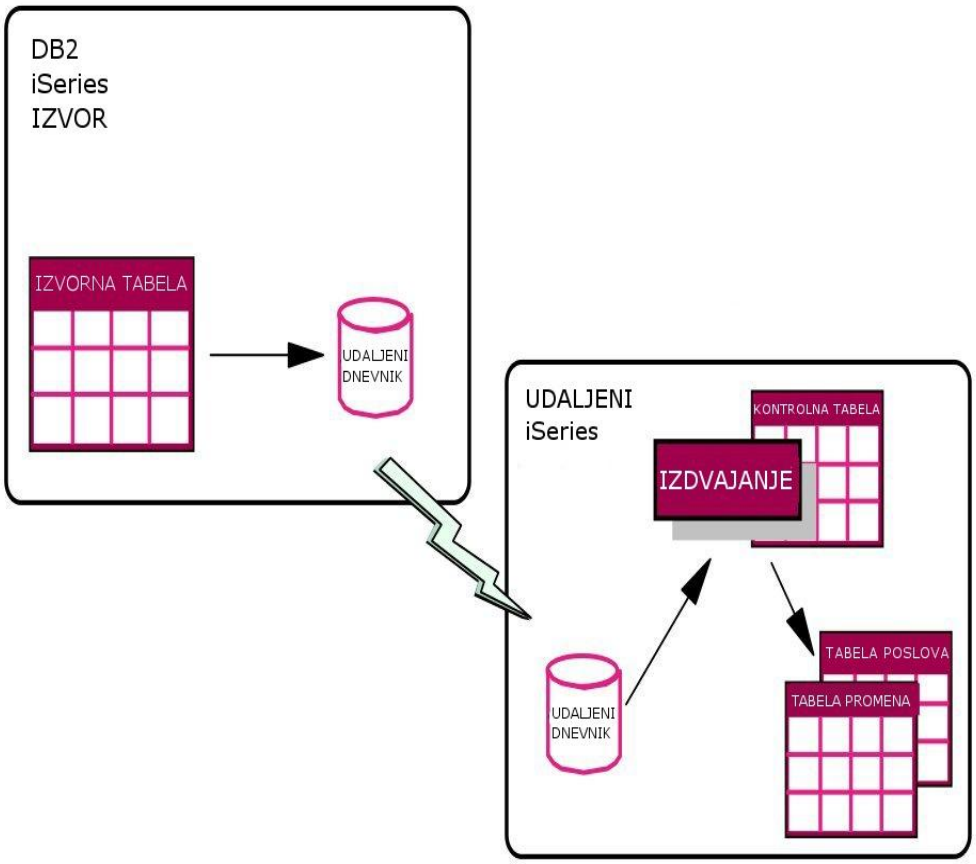
iSeries RCVJRNE command

Svaka izvorna tabela ima odgovarajuću tabelu promena (engl. *Change Data (CD)*) u kojoj se čuvaju izdvojene promene. Tabelu promena kreira centar za ponavljanje kada se definiše izvorna tabela ponavljanja. Može se izdvojiti podskup kolona izvorne tabele. Takođe može se izdvojiti vrednosti pre nego što se promene izvrše (engl. *before-image*) sa vrednostima posle izvršenja izmena (engl. *after-image*). Redni broj zapisa u dnevniku (engl. *log record sequence number* (*LSN*)) promene se koristi kao unikatni identifikator te promene.

Komponenta za izdvajanje čuva promene u memoriji dok se ne zatraži potvrđivanje transakcije koja je dovela do promena. Kada dođe do potvrđivanja transakcije komponenta za izdvajanje smešta izdvojene promene u odgovarajuću tabelu promena i čuva informacije o potvrđivanju transakcije u kontrolnoj tabeli za informacije

o poslovima (engl. *Unit of Work (UOW)*). Ukoliko se detektuje poništavanje transakcije, uklanjaju se izdvojene promene iz memorije.

Može se pokrenuti više procesa za izdvajanje na izvornom serveru kako bi se poboljšao protok. Svaki od ovih procesa ima svoju šemu za kontrolne tabele i svoj skup tabela promena. Šeme se definišu korišćenjem centra za ponavljanje kada se kreira kontrolna tabela za izdvajanje promena. Šema se precizira kada se definišu izvori i odredišta podataka i kada se pokrene komponenta za izdvajanje promena. Tabele promena i UOW tabele se uvek nalaze na serveru na kojem su i izvorne tabele. Jedini izuzetak je kada se koristi iSeries udaljeni dnevnik.



Slika 3 Izdvajanje kod iSeries sa udaljenim dnevnikom

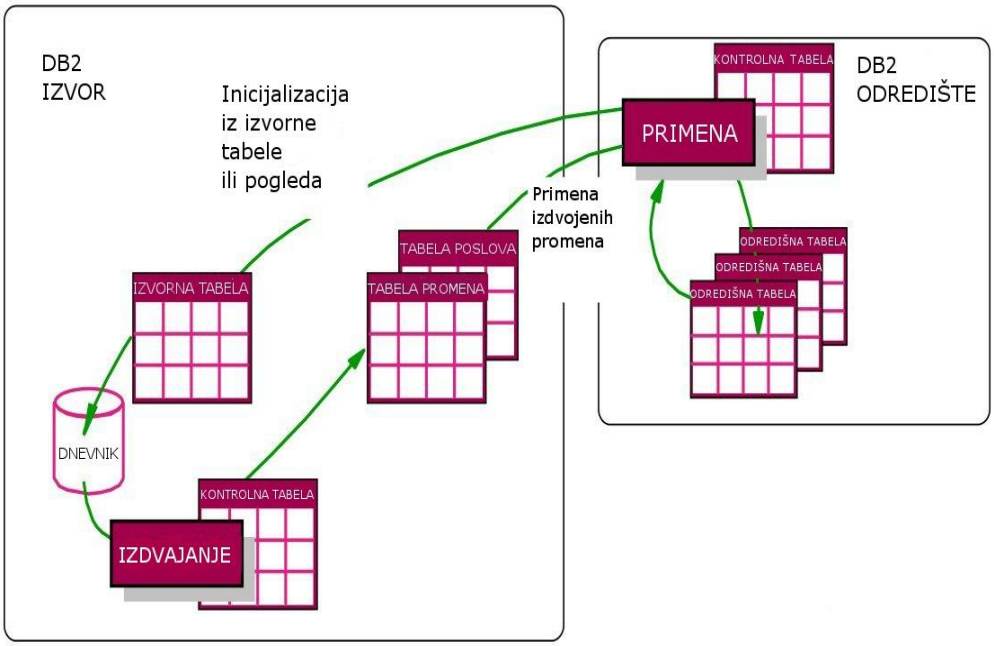
Može se podesiti udaljeno upravljanje dnevnikom na iSeries sa ADDRMTJRN komandom. Promene na izvornoj tabeli se zapisuju u lokalni dnevnik i takođe se šalju drugom iSeries sistemu sinhrono ili asinhrono. Program za izdvajanje promena može da se izvršava na udaljenom iSeries sistemu i da ponavlja izmene iz dnevnika na tom sistemu.

Za bidirekciono ponavaljanje, program za izdvajanje promena će biti pokrenut na oba servera sa programom za primenu promena pokrenutim na odredišnom serveru da može da procesira promene u oba smera. Kod ponavljanja u sistemu sa direktnim

povezivanjem (engl. *peer-to-peer*) programi za izdvajanje i primenu promena su pokrenuti na svakom serveru.

4.1.3 Primena

Izdvojene promene se primenjuju na odredišne tabele pomoću programa za primenu. Program za primenu može biti pokrenut na bilo kom serveru i mora imati vezu i ka izvornom i ka odredišnim serverima. Podaci mogu da se filtriraju po koloni, redu, spojeni sa drugim podacim (korišćenjem pogleda), i da se transformišu SQL izrazima tokom procesa primene. Bidirekciono ponavljanje, uključujući i direktno povezivanje je podržano samo za db2 familiju.



Slika 4 Primena

Centar za ponavljanje se koristi da poveže izvornu tabelu ili pogled sa odredišnom tabelom. Definiše se grupa pretplatnika (engl. *subscription set*) od jednog ili više odredišnih tabela članica (engl. *subscription members*) koje će program za primenu procesirati kao celinu. Izmene iz tabela promena se primenjuju na svaku tabelu pojedinačno, u istom redosledu u kom su nastale na izvoru. Nakon što se procesira poslednja tabela članica grupe pretplatnika izdaje se naredba za potvrđivanje. Ako odredišne tabele imaju db2 referentna ograničenja ili druge međusobne odnose koji moraju da se očuvaju, onda se mora izabrati transakciono ponavljanje pri definiciji grupe pretplatnika.

Proces za primenu promena bira inicijalni skup podataka iz izvornih tabela za prvo inicijalizovanje odredišnih tabela, korišćenjem bilo koje transformacije ili filtriranja koje je definisano. Ovo se naziva potpuno osvežavanje (engl. *full refresh)*. Postoji mogućnost da ovaj korak uradi korisnik. Centar za ponavljanje podataka pruža

mogućnost ručnog potpunog osvežavanja (engl. *Manula Full Refresh*) da bi se ažurirale kontrolne tabele procesa za primenu promena kada se inicijalno popune odredišne tabela van procesa ponavljanja podataka.

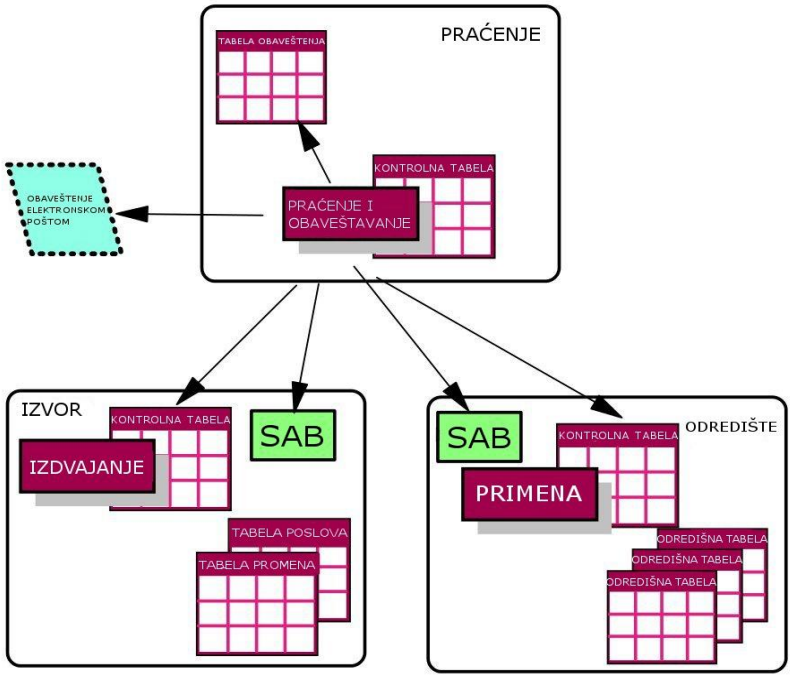
Posle potpunog osvežavanja, proces za primenu promena uzima podatke iz tabela promena i primenjuje ih na odredišnim tabelama. Neki tipovi odredišnih tabela mogu da zahtevaju spajanje sa tabelom promena i UOW tabelom tokom procesa usklađivanja promena. Spajanje nije potrebno za odredišta koja nemaju ni jednu kolonu u UOW tabeli u njenim predikatima, korišćene za distribuciju podataka i integraciju podataka. Spajanje je potrebno za bidirekcione kopije.

Proces za primenu može da se pokrene kao privremen proces ili kao zadatak koji se izvršava sve vreme. Pri definisanju grupe pretplatnika definiše se i raspored za ponavljanje. Raspored može biti vremenski orijentisan, na intervalima od 0 sekundi do jedne godine. Takođe može se zakazati pokretanje procesa za primenu korišćenjem mehanizma događaja. Događaj se imenuje, a zatim se upise slog u kontrolnu tabelu događaja procesa za primenu, ASN.IBMSNAP_SUBS_EVENT.

4.1.4 Praćenje i obaveštavanje

Komponenta za praćenje i obaveštavanje je uključena sa DB2 V8 za Windows i UNIX, i sa DB2 DataPropagator za z/OS i OS/390. Može se koristi za praćenje ponavljanja podataka na ovim platformama kao i na iSeries. Komponenta za praćenje i obaveštavanje ima svoj skup kontrolnih tabela definisan u centru za ponavljanje podataka. Administratori ponavljanja definišu pragove i događaje kroz centar za ponavljanje. Takođe mogu da se definišu grupe korisnika koje će dobijati e-mail obaveštenja.

Server na kom je pokrenut proces za praćenje i obaveštavanje se naziva server za praćenje (engl. *Monitor Server*). On može da nadgleda jedan ili više lokalnih ili udaljenih servera. Proces ne nadgleda okidače prilikom izdvajanja podataka na ne-db2 izvornim serverima.



Slika 5 Praćenje i obaveštavanje

Program za praćenje i obaveštavanje sakuplja informacije iz kontrolnih tabela komponenti za izdvajanje i primenu. On takođe koristi server za administriranje baze (engl. *Databes Administration Server (DAS)*) instaliran na serverima za primanje udaljenih komandi i slanje sistemskih informacija.

Proces za praćenje i obaveštavanje se može pokrenuti iz centra za ponavljanje podataka ili izdavanjem komande. Može se i odrediti koliko često proces proverava događaje i pragove korišćenjem parametra za interval provere.

Kada se desi nadgledani događaj, npr. poruka o grešci ili je prag prekoračen, proces za praćenje i obaveštavanje ubacuje alarm u ASN.IBMSNAP_ALERTS tabelu i šalje e-mail navedenim kontaktima.

4.2 Microsoft Replication Services

Nastariji model za distribuiranje podataka kompanije Microsoft predstavlja upravo ponavljanje podataka.

Ponavljanje podataka je zasnovano na izdavač/pretplatnik modelu (engl. *publisher/subscriber*), sa tri izdvojene uloge koje baze podataka na svakom čvoru mogu imati: izdavač, pretplatnik i distributer (engl. *publisher*, *subscriber*, *distributor*). Ove uloge funkcionišu zajedno, kako bi se iskopirao efekat izvršavanja transakcije ili stanje baze podataka sa jednog čvora na drugi. U ovom odeljku će biti predstavljena terminologija vezana za ponavljanje podataka i modele topologije.

SQL Server Replication sistem koristi koncept koji se može poistovetiti sa izdavačkom industrijom za predstavljanje komponenti sistema za ponavljanje podataka (izadavač, distributer, pretplatnici, publikacije (engl. *publications*), članci (engl. *articles*),

i pretplate (engl. *subscriptions*)). Microsoft SQL Server Replication sistem možemo posmatrati kroz koncept novinskog magazina:

- Izdavač napravi jednu ili više publikacija
- Publikacija sadrži članke
- Izdavač ili distribuira magazin direktno ili koristi distributera
- Pretplatnici primaju publikaciju na koju su se pretplatili

Iako je koncept magazina koristan za razumevanje, važno je istaći da SQL Server Replication sistem uključuje i funkcionalnost koja nije obuhvaćena ovim konceptom, a to je mogućnost da pretplatnik ažurira podatke kao i da izdavač šalje dodatne izmene na člancima publikacije.

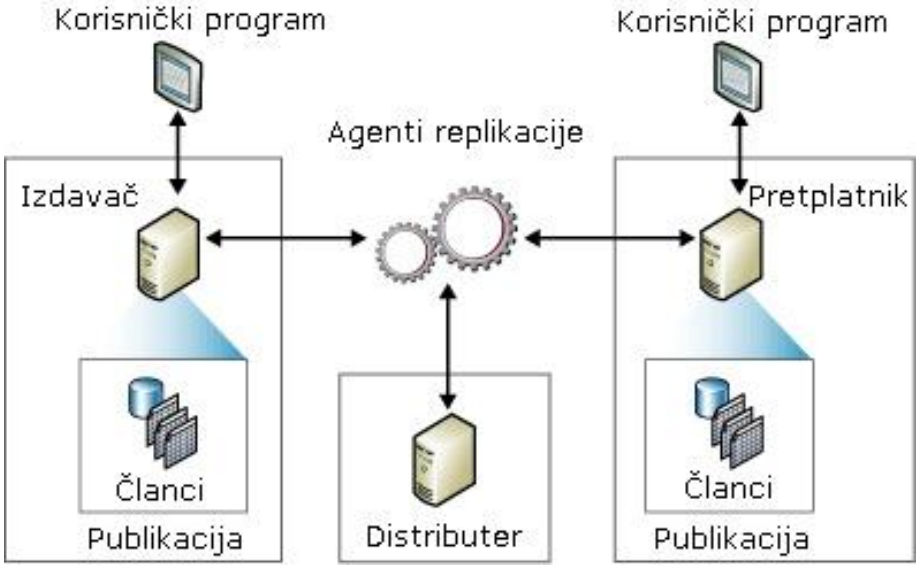
Izdavača u većini scenarija predstavlja zapravo produkciona baza podataka koja definiše skup objekata koji se ponavljaju (publikacija), prihvata sve transakcione aktivnosti i omogućava da te aktivnosti budu raspoložive i na drugim lokacijama u distribuiranom sistemu. U opštem slučaju, proizvođač skladišti u transakcionom logu transakcije koje će se prosleđivati, i to sve dok ih distributer ne izabere za prosleđivanje.

Distributer je servis koji je odgovoran za prosleđivanje podataka koji se ponavljaju. U tipičnom režimu, distributer će očitavati podatke iz loga izdavača, a zatim te podatke smeštati u sopstveni log da bi ih distribuirao pretplatniku. Radi se, zapravo, o sačuvaj i prosledi (engl. *store and forward*) servisu.

Pretplatnik prima podatke koje prosleđuje distributer. U većini situacija ova baza podataka se može samo očitavati, zato što transakciona aktivnost treba da bude vezana za izdavača. Međutim, postoje i izuzeci od ovog pravila, kao što je model sa mešovitim ažuriranjem (engl. *Merge replication*) koji dozvoljava i ažuriranje od strane pretplatnika.

Članak je pojedinačna kolekcija ponovljenih podataka koji se obično nalaze u jednoj tabeli.

Publikacija je kolekcija članaka. Pretplatnik može da se pretplati na pojedinačni članak ili na celokupnu publikaciju.



Slika 6 SQL Server Replication sistem

SQL Server 2008 podržava tri osnovne varijante sinhronizacije podataka i sve tri rade asinhrono: [3]

1. Sinhronizacija na osnovu trenutnog stanja (engl. *Snapshot Replication*)

Ovaj pristup se fokusira na kopiranje trenutnog stanja podataka jedne baze podataka u drugu bazu podataka. Celokupne stranice sa podacima se kopiraju sa jednog mesta na drugo, što znači da podaci stare od trenutka kada se formira inicijalna kopija, do trenutka kada se generiše svaka dodatna kopija. Iako je moguće definisati učestanost sinhronizacije, ovaj pristup najbolje funkcioniše kada se radi sa malim količinama podataka, koje se sporo menjaju ili se toleriše prikrivenost greške.

2. Transakciona sinhronizacija (engl. *Transactional Replication*)

Ovo je standardni model sinhronizacije. U ovom pristupu, izdavač beleži transakcije u svom logu. Periodično, servis za očitavanje loga očitava transakcije i šalje ih do distributera, koji izvršava transakcije vezane za svaku od prijavljenih baza podataka. Ovaj proces može da se izvršava prema definisanom redosledu, i to u pravilnim intervalima, ili se može konfigurisati tako da se izvršava svaki put kada se potvrde transakcije na osnovnom serveru.

3. Mešovita sinhronizacija (engl. *Merge Replication*)

Ovaj model vrednuje nezavisnost podataka, umesto konzistentnosti. Za razliku od prethodna dva modela, koji zahtevaju da se baze pretplatnice tretiraju kao da se podaci u njih mogu samo slivati, ovaj model dozvoljava promene na bazama pretplatnicama, uz mogućnost spajanja izmena koje su izvršene nad njima sa podacima koji se nalaze na izdavaču. Iako se na taj način povećava nezavisnost čvorova, povećava se i verovatnoća da se pojave problemi vezani za sinhronizaciju između baze pretplatnice i baze izdavača.

Postoje dva tipa pretplate odnosno, protokola pri sinhronizaciji podataka: [3]

Guraj (engl. *Push*) pretplata – Ovo je model pretplate u kome se distributer periodično povezuje sa bazom pretplatnicom i izvršava odgovarajuće transakcije koje su neophodne da bi pretplatnica bila ažurna.

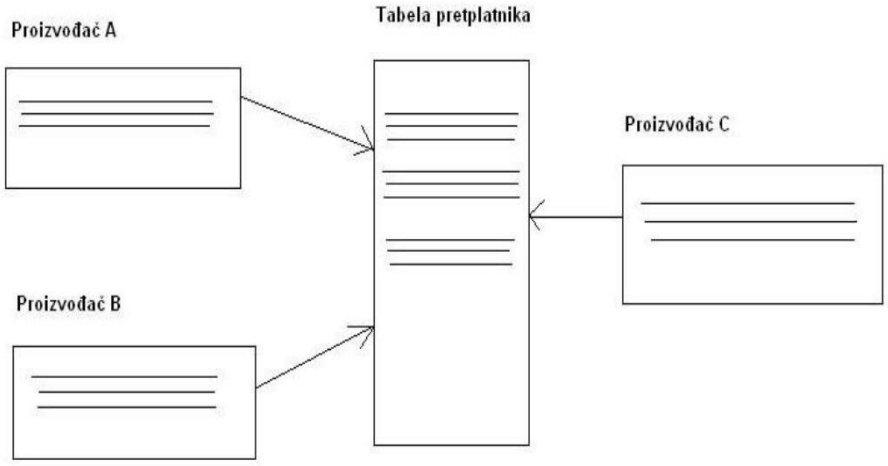
Vuci (engl. *Pull*) pretplata – Ovo je model pretplate u kome baza pretplatnica inicira sa distributera donošenje transakcije koje će izvršiti, kako bi podaci na njoj postali ažurni. Ovaj tip pretplate omogućava da se briga o sinhronizaciji pomeri sa distributera na pretplatnika, čime se mogu postići mnogo bolji rezultati kada se radi o balansiranju opterećenja u posmatranom modelu.

Pored prethodno navedenih uloga definisanih na nivou baze, tipova sinhronizacije i pretplate, postoji mogućnost kombinovanja ovih komponenti na različite načine, kako bi se definisale različite topologije.

4.2.1 Topologija

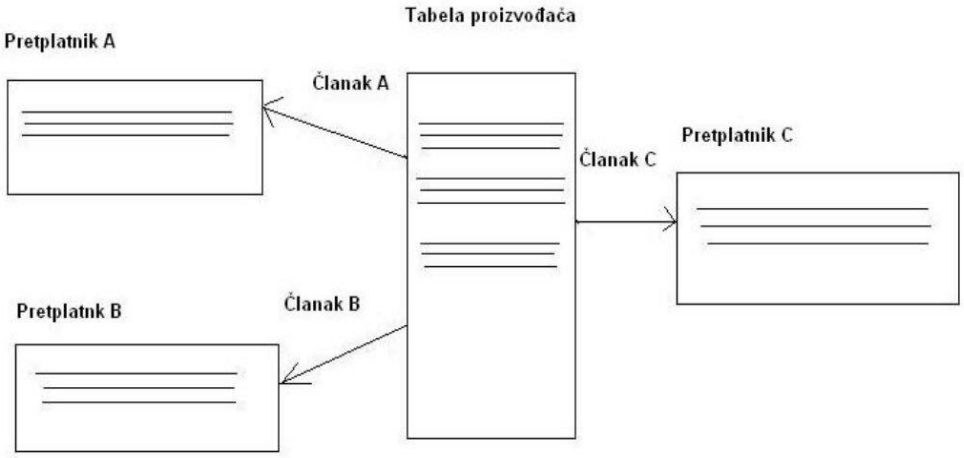
SQL Server 2008 podržava sledeće topologije distribuiranog sistema: [3]

1. Centralni pretplatnik – Postoji mogućnost postojanja više izdavača koji objavljuju podatke za jednog potrošača. Podaci za svaki članak mogu biti objavljeni i u istoj tabeli potrošača, pod uslovom da su objavljeni podaci međusobno različiti.
   Ovaj šablon, prikazan na slici 7, predstavlja najčešće primenjivani šablon kada je neophodno centralizovanje podataka za potrebe izveštavanja ili donošenja odluka.



Slika 7 Centralni pretplatnik

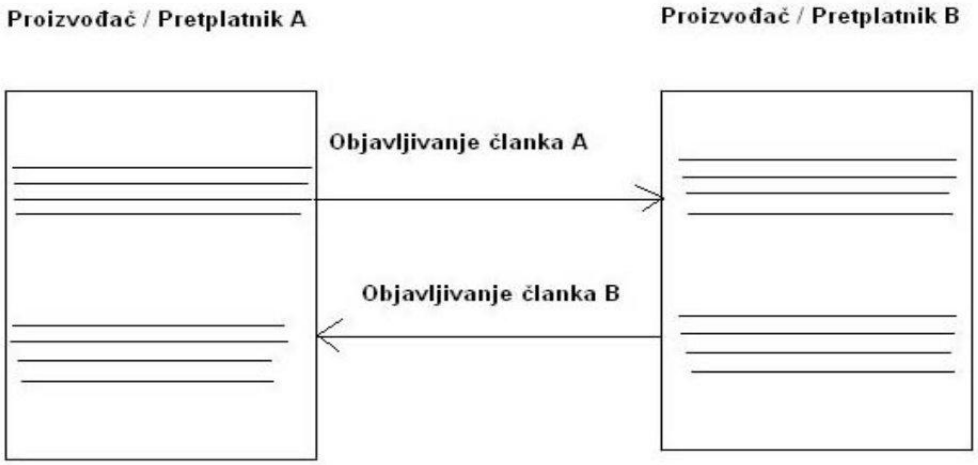
2. Centralni proizvođač – Postoji mogućnost konfigurisanja i jednog proizvođača, koji može da particioniše podatke i prosleđuje ih do različitih pretplatnika. Ukoliko pretpostavimo da svaki od pretplatnika zahteva samo podskup podataka, administrator može da kreira veći broj članaka, a svaki od njih filtrira podatke za konkretnog pretplatnika. Svaki pretplatnik može da očita podatke koji su relevantni za njega. Ovaj pristup, prikazan na slici 8, je koristan kada se podaci centralno prikupljaju, ali moraju lokalno da se distribuiraju radi decentralizovane obrade.



Slika 8 Centralni proizvođač

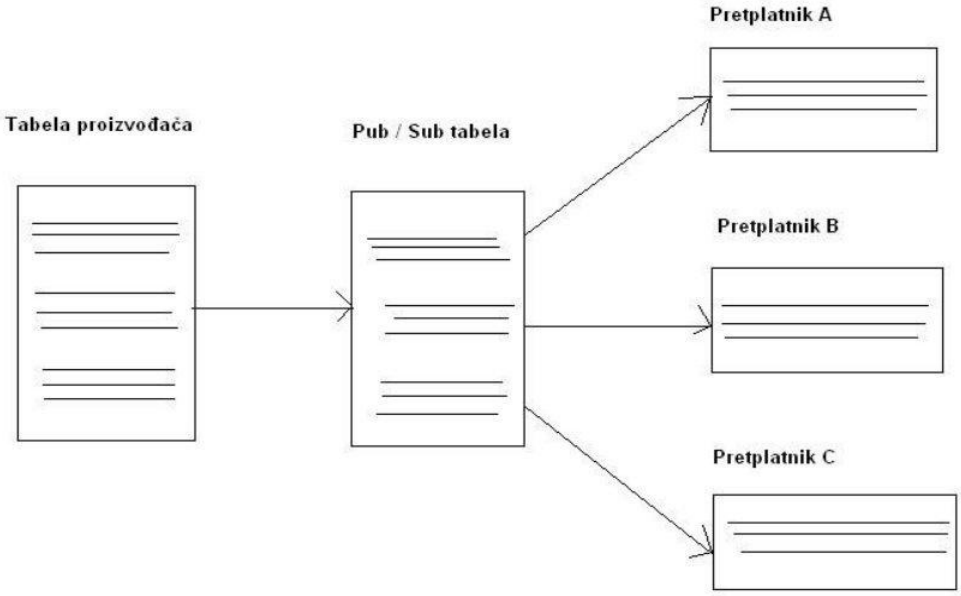
3. Regionalni proizvođači/pretplatnici – Postoji i mogućnost da svaki server treba da održava svoje podatke, ali i da šalje i prima ponovljene podatke sa drugih servera. U ovom prsitupu, koji je prikazan na slici 9, neophodno je definisati veoma decentralizovano okruženje. Međutim, jedan od problema kod ovog modela je

njegova skalabilnost. Dodavanje dodatnih čvorova u model dovodi do eksponencijalnog porasta opterećenja za svaki pojedinačni dodatni čvor.



Slika 9 Regionalni proizvođači / pretplatnici

4. Distribuirani pretplatnik/izdavač – Projektanti često primenjuju ovaj model kada je potrebno podatke ponoviti na većem broju pretplatnika, ali su pretplatnici geografski locirani tako da je neefikasno ili nepraktično da se izvršava direktna sinhronizacija. Svi podaci se sinhronizuju na odgovarajućem pretplatniku, koji ima mnogo bolji pristup većem broju pretplatnika. Nakon toga, pretplatnik ponovo izdaje podatke pretplatnicima koji ih zahtevaju. U ovom modelu, koji je prikazan na slici 10, server je odgovoran za servise pretplate i izdavanja.



Slika 10 Distribuirani pretplatnik/izdavač

Naravno, postoje i drugi šabloni koje je moguće koristiti, kao i gotovo neograničen broj kombinacija ovih šablona. Neophodno je voditi računa o performansama i pouzdanosti modela, što mora biti utvrđeno pre nego što model počne da se praktično pimenjuje. Ono što dobro izgleda na papiru, obično ne funkcioniše tako dobro u realnom sistemu.

4.3 Oracle

Baze podataka i Internet su širom sveta omogućili saradnju i razmenu podataka čineći baze podataka dostupnim u raznim organizacijama i zajednicama. Korisnici malih preduzeća, baš kao i onih globalnih koja posluju širom sveta, zahtevaju pristup podacima u svakom trenutku. Bez ovog pristupa, prihodi i klijenti mogu biti izgubljeni, pritisak može imati trajan efekat na korisnike i učiniti da ugled kompanije bude loš. Ponavljanje podataka u Oracle okruženju je proces korišćenja PL/SQL paketa i procedura za upravljanje bazama podataka, koje se isporučuju u kombinaciji sa skupom alata za mrežne programe koji treba da omoguće deljenje objekata baze podataka i podataka izmedu više baza podataka. Da bi se održavali ponovljeni objekti baze podataka i podaci između više baza podataka, promene na jednom od ovih objekata dele se sa drugim povezanim bazama podataka. Na ovaj način, objekti baze podataka i podaci se održavaju sinhronizovanim sa svim bazama koje se distribuiraju.[4]

4.3.1 Tipovi sinhronizacije

Sinhronizacija može biti automatska i manuelna. Manuelna sinhronizacija se koristi za kopiranje podataka unutar jedne baze, za kopiranje podataka između nepovezanih baza i za kopiranje podataka između baza koje ne predstavljaju "pravu" distribuiranu bazu. Neki od popularnih Oracle mehanizama za manuelnu sinhronizaciju jesu npr. export/import, prenosne tabele (engl. *transportable tablespace*) ili CTAS naredba. Može se govoriti i o sinhronizaciji temeljenoj na podacima iz dnevnika (engl. *log-based replication*) i sinhronizaciji temeljenoj na transakcijama (engl. *transactional replication*). Dnevnik (engl. *log*) je datoteka u kojoj se čuvaju radnje ažuriranja (i/ili rezultati tih radnji) koje su rađene nad podacima u bazi podataka. Ti podaci uglavnom služe bazi da uspostavi konzistentno stanje podataka nakon privremenog (npr. zbog nestanka struje) pada baze ili spašavanja podataka nakon oštećenja diska sa podacima baze. Osim svoje primarne uloge, ova datoteka se često koristi i za druge svrhe (npr. Oracle Data Guard), pa i za sinhronizaciju podataka u distribuiranom sistemu. Jedan od pionira u korišćenju dnevnika za sinhronizaciju podataka je baza Sybase. Oracle je sve do verzije 9.2 imao samo sinhronizaciju temeljenu na transakcijama, a u bazi 9.2 pojavila se i sinhronizacija temeljena na dnevniku, tj. Oracle Streams. Prednost sinhronizacije temeljene na dnevniku je njena jednostavnost i mogućnost ponavljanja i sinhronizacije daleko veće količine podataka.

Sinhronizacija podataka se još može podeliti na sinhronu i asinhronu. Kod sinhrone, proces ažuriranja kopija izvodi se u istoj transakciji kao i proces ažuriranja primarne baze. Prednost je automatska sinhronizacija podataka u svim bazama (sinhronizacija je ili svuda ili nigde uspela, što je osigurano mehanizmom dvofaznog protokola potvrđivanja), dok kod asinhrone može doći do konflikta između vrednosti

podataka na različitim bazama i te konflikte mora rešavati programer (kod programiranja aplikacije) ili administrator baze (DBA).

Nažalost, sinhrona sinhronizacija ima i veliku manu – sve baze moraju biti raspoložive da bi uspela. Oracle unutar "stare" verzije podržava i sinhronu i asinhronu varijantu, dok Oracle Streams (temeljena na dnevniku) radi isključivo asinhrono.

Takođe, može se govoriti o podeli na sekvencijalnu i proceduralnu sinhronizaciju. Kod sekvencijalne sinhronizacije promene nad kopijom podataka rade se red po red, iako su se na izvornoj bazi možda izvodile samo jednom naredbom (npr. UPDATE na izvornoj bazi može ažurirati 100 redova, ali će se na udaljenoj bazi izvesti kroz 100 UPDATE naredbi). Za razliku od toga, proceduralna sinhronizacija poziva udaljenu proceduru koja može na udaljenoj bazi kroz jednu naredbu ažurirati više redova. Napomenimo da "stara" verzija ima i sekvencijalnu i proceduralnu varijantu, a obe mogu raditi sinhrono i asinhrono. Streams radi isključivo sekvencijalno i samo asinhrono. [4]

Treba još pomenuti jednosmernu i dvosmernu sinhronizaciju. Jednosmerna je ona kod koje samo jedna baza može ažurirati određenu tabelu i slati kopije podataka drugoj bazi, a kod dvosmerne dve ili više baza mogu ažurirati istu lokalnu tabelu, pa sinhronizacija treba da ide u dva smera.

4.3.2 Oracle Streams

Oracle polaže velike nade u Oracle Streams i namenio ga je za više stvari, a ne samo kao podršku za distribuirane sisteme sa ponovljenim podacima. Oracle Streams je namenjen i za punjenje skladišta podataka (engl. *Data Warehousing*) i za upravljanje redovima poruka (engl. *Message Queuing*). Oracle Streams se pojavio u bazi 9.2, ali je u bazi 10g značajno poboljšan. Streams za distribuiran sistem sa ponavljanjem podataka podržava deljenje objekata baze podataka koji nisu identični u većini baza. Različite baze podataka u Streams okruženju mogu da sadrže deljene objekte baze podataka različite strukture. Moguće je konfigurisati pravila za transformaciju tokom zadržavanja, širenja i primene bilo koje neophodne izmene na LCRs tako da ona može biti primenjena na odredišnu bazu podataka. Oracle Streams, ugrađena funkcija u okviru Oracle baze podataka, omogućava ponavljanje i integraciju podataka. Pruža fleksibilnu infrastrukturu koja zadovoljava širok spektar potreba koje se javljaju prilikom razmene informacija. Oracle Streams omogućava širenje podataka, transakcije i događaje u okviru baze podataka ili iz jedne baze u drugu.

Oracle Streams ima fleksibilnu arhitekturu i nudi razne mogućnosti za implementaciju:[4]

• Kreiranje baze za izveštavanje i preuzimanje podataka u *oflajn* modu sa OLTP lokacije.

• Unapređuje skalabilnost i dostupnost za call centre ili slične aplikacije.

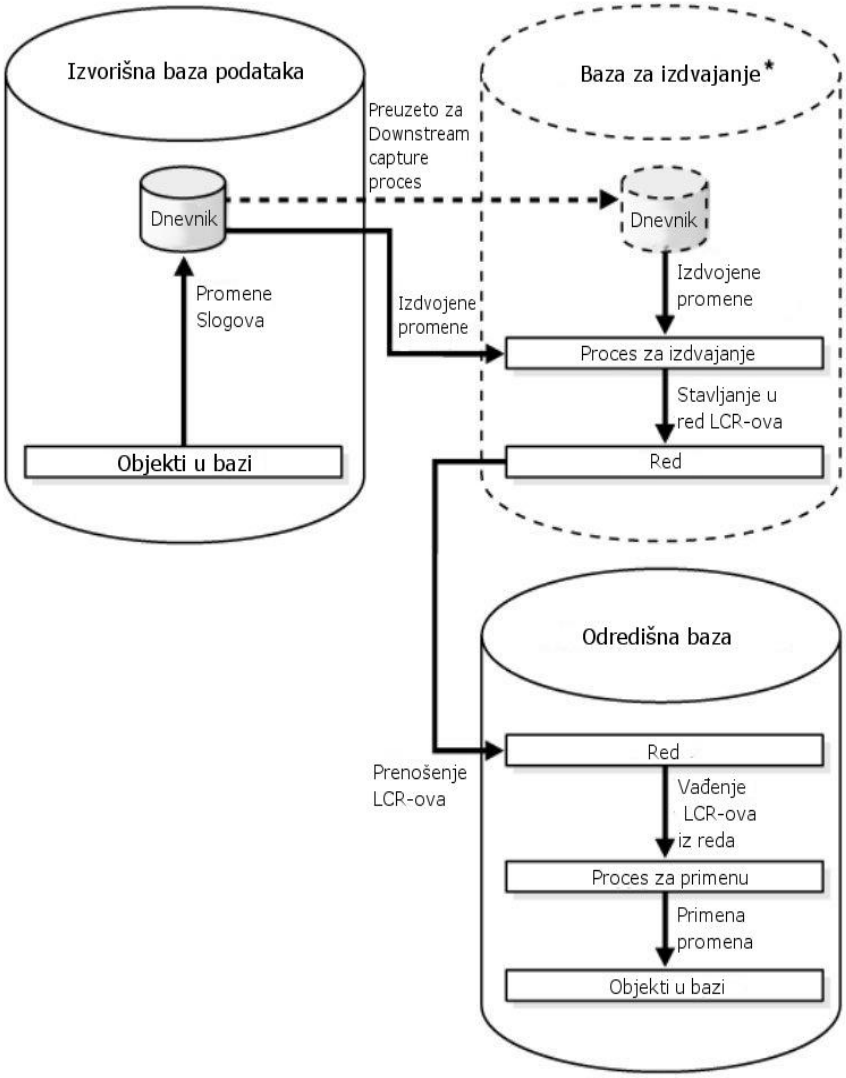
• Omogućava brz, lokalni pristup podacima sa indrektnom konekcijom, kao što su udaljeni prodajni objekti.

• Transformacija i konsolidacija podataka iz različitih lokacija, kao što su regionalna predstavništva.

• Omogućava notifikaciju događaja i tok podataka.

Oracle Streams se temelje na PL/SQL paketu DBMS_LOGMNR koji se pojavio već u bazi 8. U bazi 9.2 se zajedno sa Oracle Streams pojavio i novi PL/SQL paket

DBMS_RULE, koji služi za donošenje odluka o prihvatanju ili odbacivanju redova podataka u Streams procesima. Streams procesi su Capture, Propagation i Apply proces, tj. procesi za sakupljanje, prenošenje i primenu promena sa izvorne baze na ciljnu bazu, što se može videti na slici koja sledi.[4]



Slika 11 Arhitektura Oracle Streams

Streams radi tako da prvo na izvornoj bazi proces za sakupljanje čita promene iz *onlajn* ili arhiviranih log datoteka, stvara od toga tzv. LCR podatke (Logical Change Record) i upisuje LCR-ove u odgovarajući red koji se nalazi na izvornoj bazi. Pritom ovaj proces može koristiti "mehanizam pravila" (engl. *rules engine*), tj. može primeniti pozitivna ili negativna pravila u odluci da li neku promenu treba staviti u red. U bazi 10g osim procesa za sakupljanje postoji i Down Streams Capture, kod kojeg je moguće uzeti,

isključivo arhivirane, log datoteke sa jedne baze, prebaciti ih (na bilo koji način) na drugu bazu, a onda tamo primeniti proces za sakupljanje. Red, u koji proces za sakupljanje zapisuje, sastoji se od dela koji se nalazi u bazi, ali i od odgovarajućeg dela unutar SGA memorijskog prostora.

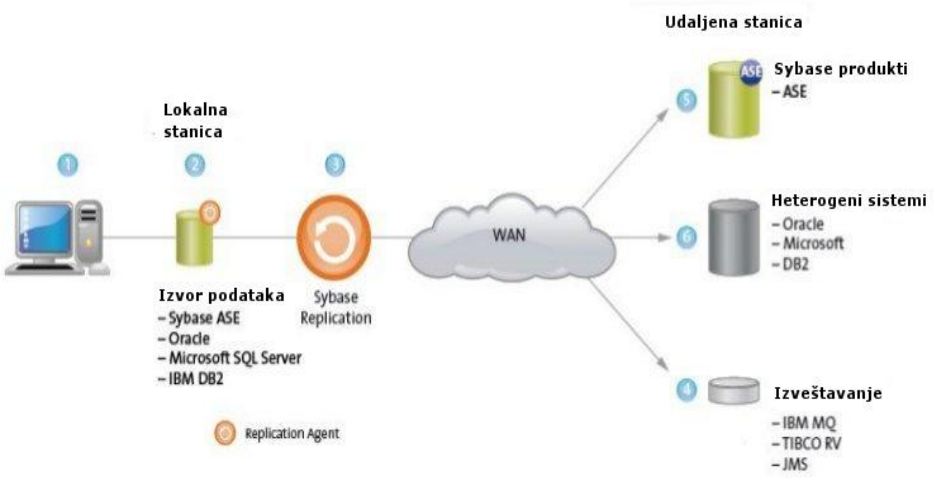
Sledeći proces za prenošenje, čita LCR podatke iz reda na izvornoj bazi, šalje ih na ciljnu bazu i upisuje u red na njoj. Ovaj proces je realizovan preko Oracle servisa za obradu poslova (engl. *job*) koji se kontinuiramo pokreće, u kratkim intervalima. Podaci iz jednog izvornog reda mogu se slati i na više ciljnih redova, bilo pomoću jednog ili više procesa za prenošenje. Proces za prenošenje, kao i proces za sakupljanje, može koristiti pravila za odlučivanje o tome koje podatke prenositi, a koje ne. Na odredišnoj bazi proces za primenu promena čita LCR-ove iz lokalnog reda i DML ili DDL naredbe nad svojom bazom. Kod procesa za primenu je moguće uz primenu pozitivnih i negativnih pravila, kao i u ostalim Streams procesima, primeniti i pravila za transformaciju podataka (engl. *Rule Based Transformation*). Moguće je napisati vlastite (PL/SQL) procedure (enlg. *Apply Handlers*), a postoje četiri vrste ovih procedura : DML, DDL, Message i Pre-Commit Handlers. Njima process za primenu može poslati LCR podatke kao parametre, a procedure dalje rade onako kako je u njima isprogramirano. Zbog svih tih mogućnosti, Streams može biti temelj za ETL (Extracting / Transforming / Loading) procese kod skladišta podataka. [16]

Kao i kod svake asinhrone sinhronizacije, tako se i ovde mogu desiti konflikti u podacima. Oni su manje verovatni nego u "staroj" *multi-master* organizaciji sistema, zato što se podaci između baza sada mogu kretati daleko većom brzinom, pa je manja šansa da do konflikata dođe. U slučaju da do konflikta dođe, Oracle nudi mehanizme za pomoć u rešavanju konflikata.

4.4 Sybase

Sybase server za ponavljanje podržava razvoj okruženja za upravljanje distribuiranim sistemom sa ponovljenim podacima i sinhronizacijom Sybase, Oracle, Mirosoft Sql Server i IBM DB2 UDB baza. Više od 18 godina, pokazao se kao izuzetno pouzdana i sofisticirana softverska tehnologija za ponavljanje podataka u okviru preduzeća.

Arhitektura Sybase sistema prikazana je na slici 12.



Slika 12 Arhitektura Sybase sistema [5]

Kao što se sa slike može zaključiti, ponavljanje podataka u heterogenom okruženju funkcioniše između sledećih baza podataka:

Izvorna baza podataka	Ciljna baza podataka
Sybase ASE	Sybase ASE
Oracle	Sybase IQ
Microsoft SQL Server	Oracle
IBM DB2 UDB	Microsoft SQL Server
IBM DB2 UDB	IBM DB2 UDB

Uz pomoć ovog sistema, postojeće aplikacije mogu da se koriste na više lokacija. Omogućeno je donošenje odluka na osnovu trenutnih (današnjih) informacija, zahvaljujući stabilnosti i pouzdanosti distribuiranog poslovnog okruženja.

Prednosti Sybase servera za ponavljanje su: [5]

1. Garantuje oporavak od katastrofe (pada) – garantuje isporuku i oporavak podataka.

2. Omogućava izveštavanje u relanom vremenu, bez uticaja na produkcijski sistem – izbegava faktore koji utiču na performanse operativne baze podataka i dobija podatke u realnom vremenu snimanjem promena.

3. Pruža efikasnu sinhronizaciju podataka i distribuciju – Podržava efikasniju i sofisticiraniju dvosmernu sinhronizaciju baze u heterogenom okruženju i na više geografskih lokacija, obezbeđujući svoje operativne podatke na raspolaganju gde i kad su potrebni.

4. Obezbeđuje neprekidnu putanju migracije – Omogućava kretanje od starijeg operativnog sistema ili baze podataka ka novoj platformi bez prekida u poslovanju.

5. Podržava heterogene baze podataka – Podržava distribuirane sisteme baza podataka sa ponovljenim podacima u realnom vremenu kroz širok spektar baza podataka, uključujući Sybase, Oracle, IBM, SQL Server.

Ključne karakteristike Sybase servera za ponavljanje su:

- Dobre performanse u pogledu distribucije i konsolidacije.
- Dvosmerna sinhronizacija u heterogenim izvorima podataka.
- Centralizovana administracija složene primene.
- Fleksibilan prevod podataka.
- Distribuirana arhitektura sa garantovanom isporukom.
- Asinhrono usklađivanje.

Performanse su diktirane brzinom Sybase servera. Ovo tvrđenje je dokazano nedavnim objavljivanjem servera koji je od 400 % do 600 % brži od svog prethodnika. Ovo povećanje performansi može se pripisati sledećim mogućnostima i funkcijama:

• Ponavljanje SQL naredbi.
• Ponavljanje uskadištenih procedura.
• Optimizacija performansi transakcija u ciljnoj bazi.

Jednom podešeno, ovo okruženje može biti automatizovano za ponavljanje podataka u skladu sa poslovnim zahtevima. Bez obzira na okruženje, bez obzira koliko je kompleksan ili široko distribuiran, bez obzira na vremenska ograničenja, Sybase server za ponavljanje može da zadovolji većinu zahteva za kretanjem podataka u okviru organizacije.

4.4.1 Tipovi ponavljanja podataka

Sybase je razvio rešenje za distribuciju podataka koje omogućava eliminisanje i smanjenje mnogih potencijalnih troškova.

Postoje tri tipa ponavljanja koja su na raspolaganju u okviru Sybase servera za ponavljanje: [8]

1. Objavi – pretplati se (engl. *Publish-and–subscribe*) model - Transakcija se dešava u izvornoj bazi podataka, a detektovana je od strane upravljača ponavljanjem (engl. *Replication Agent*) i prenosi se do lokalnog servera za ponavljanje, koji distribuira te podatke kroz LAN i WAN mrežu do servera za ponavljanje na ciljnoj bazi. Primarni podaci predstavljaju izvor podataka koje server kopira na druge baze.

2. Topla pripravnost (engl. *Warm standby/MSA)* – Server za ponavljanje obezbeđuje pripravnost uz pomoć para baza podataka; jedna služi kao aktivna baza podataka, a druga kao pasivna baza podataka, koja je podržana od strane servera za ponavljanje. Kako klijent ažurira aktivnu bazu, server kopira transakcije na bazu podataka u pripravnosti, održavajući konzistentnost između njih dve. Ukoliko aktivna baza padne iz bilo kog razloga, može se prebaciti na pripremnu bazu podataka, čineći je na taj način aktivnom i nastaviti rad uz male prekide. MSA ima sve odlike aplikacije u pripravnosti. Pored toga, MSA omogućava ponavljanje pripravnosti na više baza podataka i omogućava da se određeni objekti baze podataka ponove ili ne ponove.

3. Podrška za heterogene servere - Sybase server za ponavljanje takođe podržava ponavljanje podataka i iz ne-Sybase servera podataka. Podaci mogu biti ponovljeni na ne-Sybase servere baza podataka, kao što su Oracle, Informix, IBM DB2 i Microsoft SQL server koristeći Sybase kapiju za direktno povezivanje (engl. *DirectConnect Gateway*). Transakcije mogu biti snimljene i prosleđene iz ne-Sybase servera podataka uz pomoć Sybase upravljača ponavljanjem. Podaci takođe mogu biti ponovljeni iz ne-Sybase izvora, preko servera za ponavljanje do ne-Sybase ciljne base. Podaci se prenose

asinhrono. Vreme potrebno da se stigne do ciljne baze zavisi od veličine transakcije, nivoa aktivnosti u toj bazi, dužine lanca (jedan ili više servera preko kojih transakcija mora da prođe na putu do ciljne baze), propusnosti mreže, zauzetosti servera. Obično, na LAN-u, za male transakcije prebacivanje podataka traje oko sekund.

4.4.2 PowerDesigner i Replication Server Manager

Pored gore navedenih mogućnosti i prednosti, Sybase server za ponavljanje takođe radi sa alatima za modeliranje, razvoj i upravljanje sistemom. Uz pomoć PowerDesigner-a mogu se kreirati i snimiti metapodaci koji se koriste kako bi se što brže i fleksibilnije opisala topologija sistema sa ponovljenim podacima kao i promene do kojih dolazi. PowerDesigner takođe stvara mnoge skripte potrebne za kreiranje i definisanje logike sistema.

Replication Server Manager je moćan, ali jednostavan za korišćenje, alat za upravljanje sistemom sa ponovljenim podacima u Sybase sistemu. Daje jedan od uobičajenih metoda za administraciju i omogućava organizaciju okruženja sistema sa ponovljenim podacima na intuitivan, siguran i jeftin način.

4.5 Stanje na tržištu softvera za replikaciju podataka i očekivani trendovi

International Data Corporation **(**IDC) u svojoj studiji ispituje veličinu tržišta softvera za distribuirane sisteme sa ponovljenim podacima i skladištenje podataka u 2009. godini i predstavlja prognoze za ovo tržište u periodu od 2010. do 2014. godine.

"IDC je očekivao da će se tržište softvera za skladištenje i ponavljanje podataka skromno oporaviti u 2010. godini, rastući 3,7 % godišnje. U periodu od 2009. godine do 2014. godine očekuje se godišnja stopa rasta od 5,9 % ", kaže Steve Scully, rukovodilac istraživanja. "Tržište softvera za replikaciju podataka će biti pod uticajem brojnih tržišnih kretanja, kako se centri podataka budu kretali ka virtuelizovanoj infrastrukturi, a softveri za replikaciju nastavili dalje da se razvijaju." [6]

Takođe, IDC u svojoj studiji, tvrdi da je tržište softvera za replikaciju u jednom od najinteresantnijih perioda. Dolazi do promena na ovom milijarde dolara vrednom tržištu i prodavci se otimaju da sačuvaju svoje izvore prihoda primenom adekvatne taktike.

"Na tržište softvera za replikaciju podataka su značajno uticala nedavna ekonomska usporavanja", kaže Steve Scully. " Pored toga, kao rezultat ove dinamike, tržište počinje da napušta princip tradicionalnih apliakcija zasnovanih na host i mrežnim rešenjima." Godina 2009. je period u kom je došlo do promena udela na tržištu za nekoliko većih dobavljača softvera za ponavljanje podataka. Ono što je neizbežno jeste prodor softvera otvorenog koda za distribuirane sisteme sa ponovljenim podacima. [6]

Međutim, ono što je u poslednjih nekoliko godina ključni trend u razvoju distribuiranih sitema jeste korišćenje nerelacionih baza podataka. Naime, sa sve većim brojem aplikacija koje se pokreću i rade u okruženjima sa ogronomnim opterećenjima u smislu količine podataka i zahteva koji se obrađuju, kao što su veb servisi, potreba za većim kapacitetima:

1. može da se menja jako brzo
2. potrebni kapaciteti su najčešće mnogo veći od trenutnih.

Problem sa skalabilnošću relacionih baza podataka nastaje kada se dostigne granica u kapacitetima jednog čvora (servera). Rešenje je distribuiranje sistema na više čvorova, međutim tu složenost relacione baze počinje drastično da raste sa povećanjem broja čvorova kao i kapaciteta samih čvorova. Sve ovo je dovelo do potrebe za implementacijom novog tipa sistema baza podataka koji se usresređuje na skalabilnost po cenu drugih povoljnosti koje nosti relacioni koncept.

Novi tip sistema baza podataka se uobičajeno naziva ključ/vrednost skladište (engl. *key/value store*). Zvanični naziv ne postoji i može se naći i pod drugim nazivima npr. *document-oriented, Internet-facing, attribute-oriented* itd. u zavisnosti od specifičnog svojstva ovog novog pristupa. [18]

Relacione baze ne mogu efikasno da podrže velike sisteme, reda veličine više terabajta. Iz tog razloga su Amazon, Google, Facebook počeli da koriste nerelacione sisteme. U slučaju distribuiranih sistema, informacije se redudantno distribuiraju kroz prsten identičnih čvorova (servera). Do podataka se dolazi preko mape ključeva. Podaci su ponovljeni i asinhrono se usklađuju tako da sistem zadovoljava slab kriterijum konzistentnosti (kad-tad je konzistentan). [17]

Distribuirane relacione baze podataka i ključ/vrednost baze podataka se suštinski razlikuju, i dizajnirane su kao odgovor na različite potrebe.

5 REALIZACIJA SOPSTVENOG REŠENJA ZA SINHRONIZACIJU PODATAKA

5.1 Opis problema

Pored raznih komercijalnih rešenja, od kojih su neka opisana u prethodnom poglavlju, u nekim situacijama nije moguće koristiti mehanizame sinhronizacije koje pužaju proizvođači baza podataka, već se javlja potreba za implementiranjem vlastite sinhronizacione logike, koja treba da zadovolji specifične zahteve sistema po pitanju količine prenesenih podataka, brzine prenosa, pouzdanosti, uređaja koji su uključeni u sinhronizaciju i slično.

Jedan od projekata firme u kojoj radim je izrada i održavanje aplikacije za kompaniju sa piramidalnom strukturom, koja ima svoja predstavništva širom zemlje i regiona. Za menadžment kompanije je vrlo bitno da ima informacije o prodaji svojih proizvoda i razne druge pokazatelje prikupljene iz svojih predstavništava, a da pritom predstavništva budu što samostalnija. Ideja je da se po potrebi izvrši prikupljanje podataka u centralni čvor, kao i da se neki podaci šalju iz centralnog u ostale čvorove, a da inače, ne postoji stalna mrežna komunikacija između centralnog i ostalih čvorova. S obzirom na navedene zahteve kao i na hijerarhiju u strukturi čvorova, rešenje je realizovano kroz distribuiranu bazu sa ponovljenim podacima. S obzirom da je moguće unositi podatke sa bilo kog čvora u sistemu, kao i da ažuriranje ostalih čvorova ne mora biti sprovedeno u okviru inicijalne transakcije, korišćena je strategija lenjog distribuiranog prenošenja ažuriranja.

5.2 Cilj projekta

Za potrebe prethodno pomenutog projekta razvijeno je rešenje za sinhronizaciju podataka u navedenom distribuiranom sistemu sa ponovljenim podacima. Može se postaviti pitanje: Zašto sopstveno rešenje, ako korišćeni sistem za upravljanje bazama podataka (Microsoft SQL Server 2008) ima ugrađenu podršku za sinhornizaciju u distribuiranom sistemu sa ponovljenim podacima?

Dva su razloga:

1. Prvi razlog je finansijske prirode. SQL Server 2008 ima tri paketa - Express, Standard i Enterprise Edition. Podrška za sinhronizaciju podataka u distribuiranom sistemu sa ponovljenim podacima je omogućena tek sa Standard verzijom, a najveći broj baza kod naših korisnika je SQL Server Express Edition.

2. Drugo, rešenje koje nudi SQL Server Standard Edition je za naše potrebe suviše zatvoreno. Obzirom da se aplikacija još uvek razvija i da se očekuju izmene u strukturi baze, više nam odgovara fleksibilnije rešenje na koje ćemo moći samostalno da utičemo.

5.3 Korišćena tehnologija

Aplikacija je razvijena u .Net okruženju i tom prilikom korišćena je tehnologija Microsoft Synchronization Services, koja je deo Sync Framework-a. Sync Framework je

platforma za sinhronizaciju podataka koja podržava bilo koji tip podataka, bilo koje skladište podataka i bilo koju topologiju mreže. Ova tehnologija prvenstveno pogoduje programerima, za razliku od klasičnih rešenja za sinhronizaciju koja su uglavnom namenjena administratorima sistema.

Praćenje promena nastalih u bazi omogućeno je upotrebom tehnologije za praćenje izmena (engl. *Change Tracking*), koja je novina u SQL Server-u 2008. Change tracking omogućava izdvajanje izmena koje su nastale nad tabelama određenog korisnika u zadatom vremenskom periodu, zajedno sa informacijama o tim promenama. Sve to olakšava praćenje promena nastalih u bazi, što je ključni element za sinhronizaciju podataka u distribuiranom sistemu.

5.3.1 Microsoft Synchronization Services

Microsoft Sync Framework je platforma za sinhronizaciju podataka. Sync Framework ima arhitekturu koja ne zavisi od specifičnog transportnog protokola i u koju mogu biti uključeni specifični sinhronizacioni provajderi implementirani kroz ADO.NET API. Sync Framework se može koristiti za *oflajn* pristup podacima, pri čemu se lokalno radi sa poslednjom prevučenom verzijom podataka, a promene se u paketima šalju udaljenoj server bazi, za sinhornizaciju promena na svim čvorovima u mreži, kao i za sinhronizaciju većeg broja direktno povezanih (engl. *peer-to-peer*) čvorova. Sync Framework ima ugrađenu podršku za otkrivanje konflikta korišćenjem oznaka konfliktnih podataka pri čemu korisnik naknadno razrešava konflikt (podaci su već ažurirani) ili primenom definisane politike za rešavanje konflikta. Sync Services uključuje i ugrađenu bazu SQL Server Compact za čuvanje meta podataka o procesu sinhronizacije.

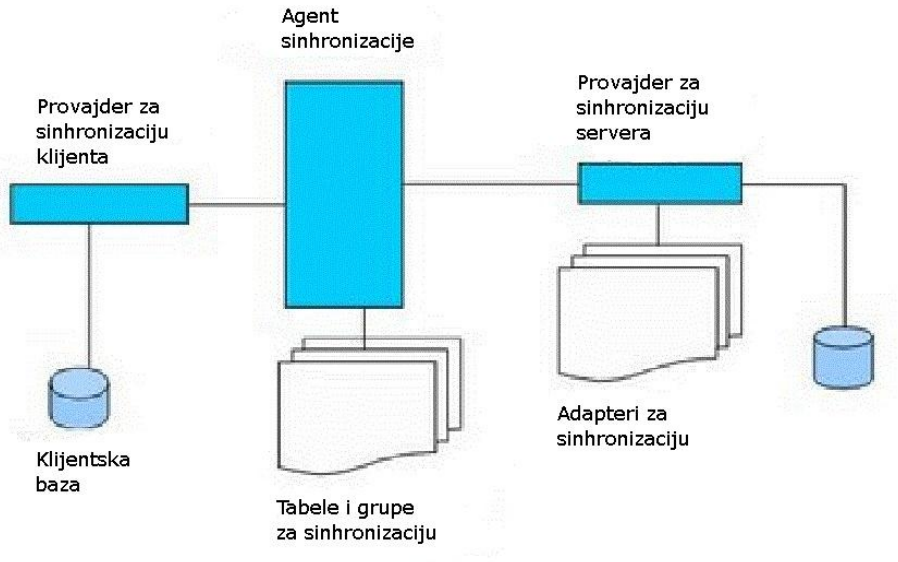
Sync Framework omogućava sinhronizaciju bez obzira na specifično skladište podataka kao i na transportni protokol. Kroz specifične sinhronizacione provajdere (engl. *synchronization providers*), bilo koji izvor podataka može biti podržan. Postoje tri sinhronizaciona provajdera: *Microsoft Sync Services for ADO.NET, Sync Services for File Systems,* i *Sync Services for SSE*.

Sync Services API kreira sinhronizacionu sesiju, (*Session* objekat). Sinhronizacija se u sesiji odvija korišćenjem dva sinhronizaciona provajdera – jedan za izvorišno skladište podataka i jedan za odredišno skladište podataka. Tokom procesa sinhronizacije odredišni provajder šalje skup podataka, izvorišni provajder upoređuje ovaj skup sa skupom promena na izvorištu i dobijene razlike šalje odredištu. Odredišni provajder ispituje da li postoje konflikti i ažurira podatke na odredištu.

5.3.1.1 Sync Services for ADO.NET

Microsoft Sync Services for ADO.NET je sinhronizacioni provajder koji omogućava sinhronizaciju baza podataka korišćenjem ADO.NET-a. Sinhronizacija se vrši nad ADO.NET skupovima podataka (engl. *DataSet*). Podržane su i druge baze podataka sem relacionih kao na primer XML baze ili web servisi.

Arhitektura Sync Services je prikazana na slici, koja sledi: [10]



Slika 13 Arhitektura SynServices

Sem dve baze, svi elementi prikazani na prethodnoj slici odgovaraju klasama Synchronization Services-a koje su sadržane u sledećim bibliotekama:

- Microsoft.Synchronization.Data.dll sadrži Synchronization Agent, SynchronizationTables, i Synchronization Groups.
- Microsoft. Synchronization.Data.SqlServerCe.dll sadrži Client Synchronization Provider.
- Microsoft. Synchronization.Data.Server.dll sadrži Server Synchronization Provider i Synchronization Adapters.

SyncAgent

Predstavlja upravljača sinhronizacijom. Obično se nalazi na strani lokalne baze i pokreće se na zahtev korisničke aplikacije. Agent koristi dva provajdera sinhronizacije, jednog za lokalnu bazu podataka i drugog za udaljenu bazu podataka.

SyncAgent upravlja procesom sinhronizacije na sledeći način:

- Prolazi kroz sve tabele koje treba sinhronizovati.
- Poziva klijentskog provajdera sinhronizacije, preuzima promene nastale na lokalnoj bazi i primenjuje promene na lokalnu bazu.
- Poziva serverskog provajdera sinhronizacije, preuzima promene nastale na udaljenoj bazi i primenjuje promene na udaljenu bazu.

Agent sinhronizacije takođe upravlja i informacijama koje se tiču sesije, kao i izveštavanjem klijentske aplikacije o uspehu sinhronizacije, greškama i statistikama.

SyncTable i SyncGroup

SyncTable se definiše za svaku tabelu koja učestvuje u sinhronizaciji. Sadrži informacije kao što je na primer, smer sinhronizacije (Snapshot, Downloadonly,

Uploadonly ili Bidirectional). Objekat SyncTable omogućava određivanje opcije za kreiranje tabele, ako je potrebno, agent može preuzeti definicije šema sa udaljenog servera i primeniti ih na lokalnu bazu podataka.

Pošto se definišu, tabele koje će se sinhronizovati mogu se dodati u sinhronizacionu grupu. Sinhronizaciona grupa je mehanizam koji obezbeđuje da ukoliko su tabele uključene u sinhronizacionu grupu, operacije koje treba izvršiti nad njima budu prenete u jednom paketu i izvršene u jednoj transakciji. Ako bilo koja od operacija paketa padne, pada cela transakcija i odlaže se za sledeću sinhronizaciju.

ServerSyncProvider

ServerSyncProvider nasleđuje klasu DbServerSyncProvider, koja je takođe deo Sync Services-a, i služi za komunikaciju sa bazom podataka i odvajanje agenta sinhronizacije od specifične implementacije baze podataka.

Provajder ima dve ugrađene komande, čijim izvršenjem se dolazi do dva neophodna podataka prilikom procesa sinhronizacije: [7]

- SelectNewAnchorCommand - Provajder koristi ovu komandu kako bi dobio oznaku nove verzije sinhronizacije iz baze (o oznakama verzije sinhornizacije će biti više reči u posebnom odeljku).
- SelectClientIdCommand – Pruža informaciju o tome sa kog čvora dolaze izmene nad podacima. U bazi na svakom čvoru postoji tabela u kojoj se čuva jedinstveni identifikator za tu bazu, u vidu GUID-a.

Osnovne aktivnosti ServerSyncProvider-a su:

- Čuvanje informacija o tabelama na serveru koje su označene za sinhronizaciju.
- Omogućavanje aplikaciji da dođe do promena koje su nastale na serverskoj bazi od poslednje sinhronizacije.
- Zadavanje transakcija kojima se stanje serverske baze usklađuje sa klijentskom bazom.
- Detekcija konflikta.

ClientSyncProvider

Omogućava pristup podacima koji se skladište na lokalnoj bazi i omogućava usklađivanje sa serverskom bazom.

SyncAdapter

Modelovan je po ugledu na DataAdapter, ali ima predefinisan skup komandi za proces sinhronizacije podataka:

- SelectIncrementalInsertsCommand – Rezultat ove komande su novi slogovi u bazi nastali od prethodne sinhronizacije.
- SelectIncrementalUpdatesCommand- Rezultat ove komande su ažurirani slogovi u bazi nastali od prethodne sinhronizacije.
- SelectIncrementalDeletesCommand – Rezultat ove komande su slogovi izbrisani iz baze od poslednje sinhronizacije.

- InsertCommand, UpdateCommand, and DeleteCommand- Komande omogućavaju unošenje, ažuriranje ili brisanje podataka koji su označeni kao uneti, izmenjeni ili obrisani na nekom drugom čvoru. Slogovi koji su novi, izmenjeni ili obrisani na nekom drugom čvoru se dobijaju komandama SelectIncrementalInsertsCommand, SelectIncrementalUpdateCommand ili SelectIncrementalDeleteCommand.
- SelectConflictUpdatedRowsCommand, SelectConflictDeletedRowsCommand – Komande omogućavaju identifikaciju onih slogova, pri čijem unošenju, ažuriranju ili brisanju dolazi do konflikta.

Konfigurisanje sinhronizacije podrazumeva sledeće korake: [14]

1. Kreiranje tabele za praćenje, za skladištenje metapodataka i kreiranje procedura za ažuriranje podataka i metapodataka u serverskoj bazi.

2. Inicijalizacija svake baze podataka šemama, podacima i praćenjem promena infrastrukture.

Izvršenje sinhronizacije podrazumeva: [15]

1. Kreiranje servera, adaptera, serverskog i klijentskog provajdera za sinhronizaciju.

2. Inizijalizacija svakog klijenta baze podataka.

3. Kreiranje upravljača sinhronizacije i sinhronizacija čvorova.

Inicijalizacija baze podataka podrazumeva kopiranje na svaku bazu podataka tabele šeme praćenja i infrastrukturu praćenja promena, kao i svih polaznih podataka koji su potrebni. Za baze podataka koje se sinhronizuju pomoću DbSyncProvidera Sync Framework ne može automatski napraviti tabelu ili šemu za praćenje promena u svim bazama podataka. Aplikacija mora da obezbedi da ovi objekti postoje pre nego što pokuša da sinhronizuje čvorove.

5.3.2 Change Tracker

U rešenjima koja se primenjuju u praksi, praćenje promena nastalih u bazi se često obavlja uz pomoć dodatnih kolona, trigera i tabela u koje se smeštaju informacije o promenama nastalim na slogovima, kao i o tipu promene (insert, update, delete). Nedostaci ovakvog pristupa su: [9]

- Promena šeme baze podataka dovodi do promena i u strukturi ovih objekata.
- Trigeri se okidaju kad god se desi neka promena u bazi što negativno utiče na performanse.
- Logika održavanja ispravne verzije i brisanja podataka može da bude veoma složena.

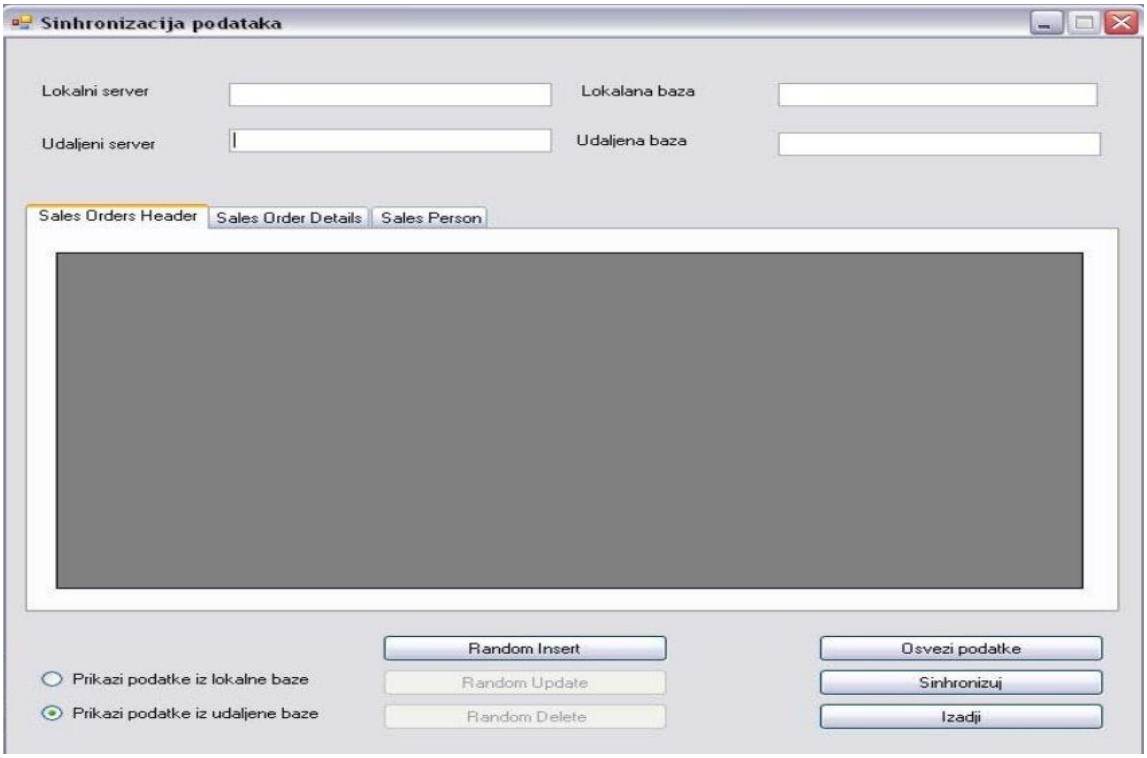
SQL Server 2008 omogućava praćenje ovih promena na jednostavan način. Praćenje promena je omogućeno na nivou tabele, SQL Server engine pamti informacije o promenama koje su nastale nad podacima tih tabela. Aplikaciji koja je integrisana sa ovom opcijom SQL Servera omogućeno je da utvrdi koji su slogovi izmenjeni i da dođe do informacija o izmenama.

Ključne prednosti Change Tracker-a su: [10]

- Promene se beleže u trenutku kada se izvrši neka DML operacija.
- Postoji funkcija koja kao rezultat vraća promene u redosledu kojim su se dešavale nad tabelom i verziju u kojoj je do njih došlo. Ova funkcija obezbeđuje pouzdane i lako čitljive podatke.
- Automatsko uklanjanje rezultata praćenja.

5.4 Specifikacija

U nastavku rada biće prikazana jednostavna aplikacija koja treba da posluži za demonstriranje razvoja servisa za sinhronizaciju podataka. Sistem je projektovan tako da se u centrali prikupljaju podaci sa svih strana - prenose se novo uneti podaci, njihove promene i brisanja. Korisniku je u ovom primeru omogućeno da pristupi lokalnoj i udaljenoj bazi, da izvrši promene i na jednoj i na drugoj bazi i da izvrši sinhronizaciju njihovih podataka. Radi bolje ilustracije onoga što se tokom sinhronizacije dešava, korisničkim interfejsom aplikacije je omogućeno da se unose, menjaju i brišu podaci na izabranoj bazi, izvrši sinhronizacija i vide rezultati nakon izvršene sinhronizacije.



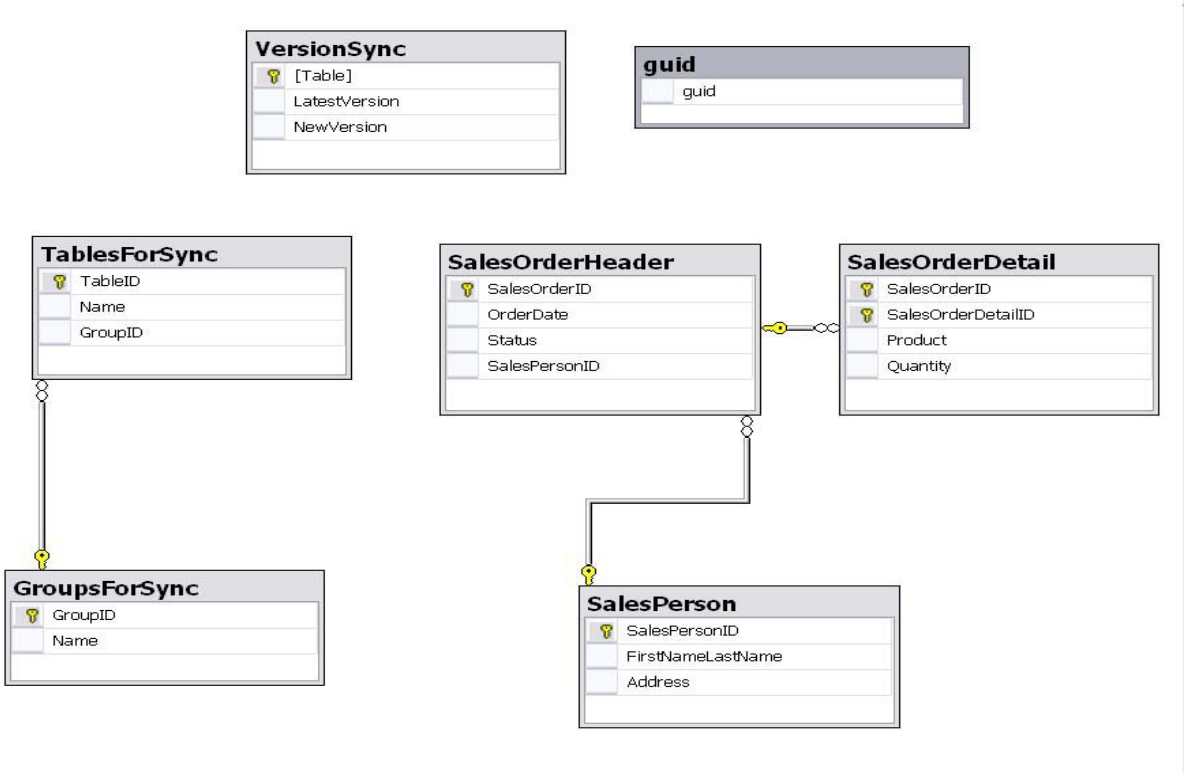
Primer 1 Forma za sinhronizaciju podataka

5.4.1 Šeme baza podataka

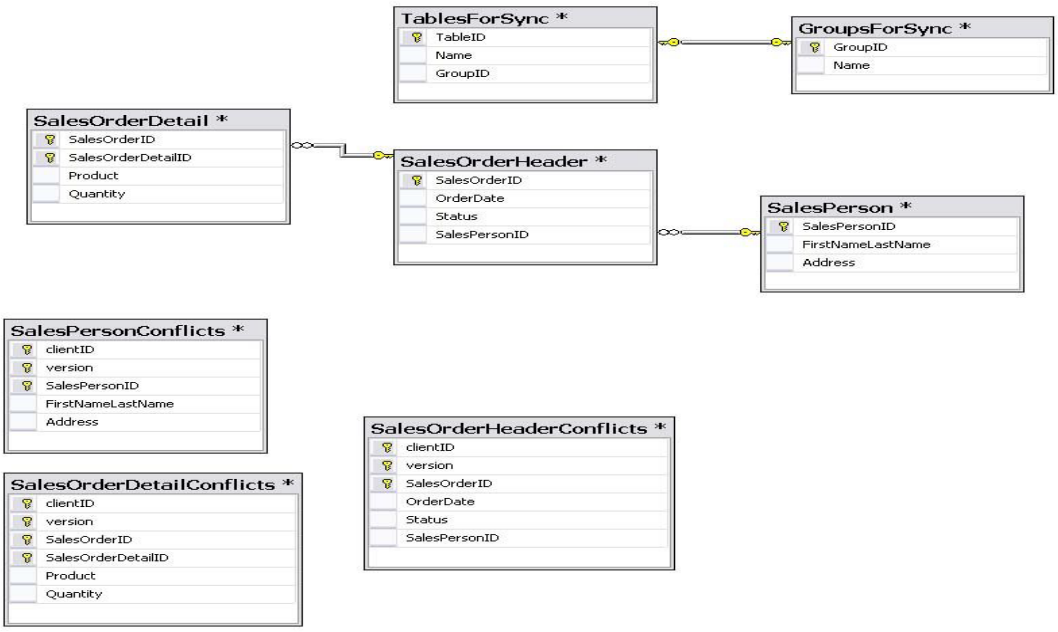
U ovom primeru, za potrebe ilustracije mehanizma za sinhronozaciju podataka, šeme baza podataka (lokalne i udaljene) će biti pojednostavljene i činiće ih samo tri tabele čiji će podaci biti sinhronizovani. Pored njih postoji još nekoliko tabela, koje služe

za konfigurisanje same sinhronizacije, tako što čuvaju podatke o verziji sinhronizacije i tabelama koje u tom procesu učestvuju.

U udaljenoj (engl. *remote*) bazi za svaku od tabela koja učestvuje u sinhronizaciji postoji tabela u kojoj se beleže konflikti, ukoliko do njih dođe. U svakoj od tabela za konflikte se beleže podaci iz tabele iz koje potiče konfliktni slog, ali i identifikator baze sa kog taj slog dolazi i verzija sinhronizacije u kojoj se konflikt javio. O konfliktima među podacima će biti više reči kasnije.



Primer 2 Šema lokalne baze podataka



Primer 3 Šema udaljene baze podataka

Relacioni model baze podataka sastoji se od sledećih relacijskih šema:
SalesOrderHeader(SalesOrderID, OrderDate, Status, SalesPersonID)
SalesOrderDetail(SalesOrderID,SalesOrderDetailID,Product,Quantity)
SalesPerson (SalesPersonID, FirstNameLastName,Address)
VersionSync (Tabel, LatestVersion, NewVersion)
GroupsForSync (GroupID, Name)
TabelesForSync (TableID, Name, GroupID)
SalesOrderHeaderConflicts (clientID,version,SalesOrderID, OrderDate, Status, SalesPersonID)
SalesOrderDetail (clientID,version,SalesOrderID,SalesOrderDetailID,Product,Quantity)
SalesPerson (clientID,version,SalesPersonID, FirstNameLastName,Address)

5.5 Implementacija rešenja

U narednoj sekciji će biti opisan način praćenja promena i kako se upiti za praćenje promena koriste da se utvrdi koje promene podataka preuzeti sa baze. Za potrebe ove aplikacije napisane su procedure koje izdvajaju promene nad podacima koji učestvuju u procesu sinhronizacije.

5.5.1 Praćenje promena u bazi

Korišćenje servisa Change tracker za praćenje promena je omogućeno na nivou servera, ali je potrebno eksplicitno omogućiti korišćenje na nivou svake baze i tabele koju želimo da pratimo. Aktivnost Change tracker-a pokreće se sledećim komandama:

```
ALTER DATABASE Local
SET CHANGE_TRACKING = ON
(CHANGE_RETENTION = 10 DAYS, AUTO_CLEANUP = ON)
ALTER TABLE Local.dbo. SalesOrderHeader
ENABLE CHANGE_TRACKING
```

Jedna od opcija prilikom omogućavanja praćenja promena na nivou baze je i to koliko dugo će meta podaci biti pamćeni i da li ih treba automatski izbrisati. Vrlo je bitno voditi računa o učestalosti sinhronizacije prilikom postavljanja ove opcije, da ne bi došlo do gubitka informacija o promenama do kojih dolazi između dve sinhronizacije.

5.5.2 Utvrđivanje promena koje treba sinhronizovati

Nakon što je omogućeno praćenje promena, u aplikaciji se uz pomoć change tracking funkcije i oznaka tekuće verzije sinhronizacije – sidra (engl. *anchor*) utvrđuje šta treba uneti, ažurirati ili obrisati da bi baze bile usklađene u pogledu podataka. Sidro se može definisati kao specijalna oznaka, koja se koristi da bi se definisao skup promena koje treba sinhronizovati.

Change tracking funkcija kao parametar ima naziv tabele čije se promene posmatraju i sidro (trenutak od kog se posmatraju promene). Nakon izvršenja bilo koje DML naredbe u bazi verzija promena Change tracker-a se povećava i moguće ju je očitati pozivom funkcije change_tracking_current_version().

Primer upita nad funkcijom promena Change tracker-a bi mogao da izgleda ovako:

```
SELECT * FROM CHANGETABLE(CHANGES SalesOrderHeader,15) CT
```

Ako je trenutna verzija u posmatranoj tabeli npr. 23 onda će ova funkcija vratiti sve one promene čija je verzija nastanka veća od 15 i manja ili jednaka 23.

Tabela rezultata ove funkcije ima sledeće kolone:

- SYS_CHANGE_VERSION – ovo polje daje informaciju o verziji u kojoj je došlo do promene nad posmaranim slogom.
- SYS_CHANGE_CREATION_VERSION – verzija u kojoj je slog kreiran.
- SYS_CHANGE_OPERATION – operacija koja je izvršena nad slogom i može uzeti jednu od vrednosti iz skupa (I,U,D).
- SYS_CHANGE_COLUMNS – informacija o koloni nad kojom je došlo do izmene.
- SYS_CHANGE_CONTEXT – u trenutku unošenja, ažuriranja ili brisanja nekog sloga u ovo polje se upisuje jedinstveni identfikator baze u kojoj se neka od navedenih operacija izvršava.
- Ključno polje/polja za tabelu čije se promene prate

Postavljanjem upita nad ovim rezultujućim skupom podataka može se doći do potrebnih informacija o tome koji podaci su uneti, ažurirani ili obrisani u periodu između dve sinhronizacije.

Za svaku od tabela postoji procedura koja kao rezultat vraća sve unete, ažurirane i izbrisane slogove.

Na primeru procedure za dobijanje unetih vrednosti između dve sinhronizacije nad tabelom SalesOrderHeader biće objašnjena logika dolaženja do podataka, koja je identična za sve ostale tabele.

```
CREATE PROCEDURE [dbo].[spSelectInsertSalesOrderHeader]
    @sync_new_received_anchor binary(2048),
    @sync_client_id_binary binary(128)
```

Svaka od procedura je identične strukture. Jedina razlika je u nazivu tabele koja se pojavljuje u nazivu procedure, a zatim i prosleđuje proceduri spSelectInsert kao jedan od parametara. Kao parametri, proceduri se prosleđuju i @sync_new_received_anchor, koji označava verziju tekuće sinhronizacije i @sync_client_id_binary, koji jedinstveno identifikuje bazu nad kojom se podaci selektuju.

Procedura spSelectInsert je napisana dinamički tako da je univerzalna za sve tabele sa kojima se radi.

```
CREATE PROCEDURE [dbo].[spSelectInsert]
    @tableName nvarchar(50),
    @sync_new_received_anchor binary(2048),
    @sync_client_id_binary binary(128)
```

U okviru procedure spSelectInsert se poziva i procedura spCreateTrackerTable koja kreira i puni privremenu tabelu podacima koje vraća change tracking funkcija za zadati kriterijum.

Parametri ove funkcije su naziv tabele, verzija prethodne sinhronizacije (@sync_last_received_anchor), verzija trenutne sinhronizacije (@sync_new_received_anchor) i operacija koja se u datom slučaju posmatra.

```
CREATE PROCEDURE [dbo].[spCreateTrackerTable]
    @tableName nvarchar(100),
    @sync_last_received_anchor int,
    @sync_new_received_anchor int,
    @sync_change_operation char(1)
```

Procedura koja unosi podatke na udaljenoj bazi se od procedure za unos podataka na lokalnoj bazi razlikuje samo po tome što detektuje konflikte i poziva proceduru koja te slogove smešta u odogvarajuće tabele. Iz tog razloga biće prikazana procedura koja unosi podatke u tabelu SalesOrderHeader i to na udaljenoj bazi.

```
CREATE PROCEDURE [dbo].[spInsertSalesOrderHeader]
    @sync_client_id_binary binary(128),
    @sync_row_count int OUT,
    @order_id int,
    @order_date datetime,
    @unit_id int
```

Obzirom na to da je primer pojednostavljen i da će u realnom sistemu biti mnogo više objekata koje treba sinhronizovati, a da za svaku tabelu koja učestvuje u sinhronizaciji treba da postoje i procedure za izdvajanje promena, kao i ažuriranje podataka u tabeli, količina objekata u bazi bi u realnom sistemu bila nedopustivo velika što bi imalo velikog uticaja na održavanje sistema i performanse. Iz tog razolga kreirane su procedure koje generišu potrebne procedure za svaku tabelu za potrebe sinhronizacije i procedure koje po završetku sinhronizacije uklanjaju ove objekte iz baze.

```
CREATE PROCEDURE [dbo].[spGenerateInsert]
    @tabeleName nvarchar(50)
```

5.5.3 Realizacija servisa za sinhronizaciju

Servis za sinhronizaciju razvijen je na programskom jeziku C#, u okruženju Microsoft .NET.

Implementacija SynAgent-a

Agent SynAgent implementiran je tako da prihvata dva provajdera sinhronizacije, jednog za lokalnu bazu podataka i drugog za udaljenu bazu podataka. Dodeljuju mu se i tabele čiji će se podaci usklađivati, a njihov spisak postoji na provajderu udaljene baze.

```
RemoteSyncProvider remoteSyncProvider = new RemoteSyncProvider ();
LocalSyncProvider localSyncProvider = new LocalSyncProvider ();
SyncAgent syncAgent = new SyncAgent();
syncAgent.RemoteProvider = serverSyncProvider;
syncAgent.LocalProvider = clientSyncProvider
foreach(SyncTable t in remoteSyncProvider.Tables)
{
syncAgent.Configuration.SyncTables.Add(t);
}
```

Implementacija SyncTable-a

Na osnovu podataka koji postoje u bazi o tabelama čiji se podaci sinhronizuju, puni se skup podataka (engl. *DataSet*) koji će poslužiti za iščitavanje podataka za podešavanje svakog SyncTable objekta.

Podaci koje treba definisati za svaku tabelu su naziv, način kreiranja, pravac sinhronizacije i naziv grupe kojoj pripada.

```
public void readTables()
{
SqlDataAdapter ldataAdapter = new SqlDataAdapter();
DataSet ldataSet = new DataSet();
string lCommand = "SELECT tableName, groupName from
vwTablsForSync";
```

```
ldataAdapter.SelectCommand = new SqlCommand(lCommand,
(SqlConnection)this.Connection);
ldataAdapter.Fill(ldataSet);
foreach (DataRow r in ldataSet.Tables[0].Rows)
{
SyncTable s = new SyncTable(r["tableName"].ToString());
s.CreationOption = TableCreationOption.DropExistingOrCreateNewTable;
s.SyncDirection = SyncDirection.Bidirectional;
s.SyncGroup = new SyncGroup(r["groupName"].ToString());
Tables.Add(s);
}
}
```

Implementacija SyncAdapter-a

Implementacija SyncAdapter-a sastoji se, u suštini, u definisanju njegovih specifičnih komandi koje zatim treba izvršiti za svaku tabelu koja učestvuje u sinhronizaciji.

Implementacija RemoteProvider-a

Implementacija RemoteProvider-a podrazumeva kreiranje konekcije na udaljenu bazu, kreiranje adaptera za svaku tabelu koja je označena za sinhronizaciju, i definicija komande kojom se iz baze čita vrednost poslednje sinhronizacije udaljene baze.

Implementacija LocalProvider-a

Većina metoda implementira se na isti način kao i u klasi RemoteProvider, s tom razlikom da LocalProvider ima metodu za ažuriranje verzije sinhronizacije koja se beleži u tabeli VersionSync na lokalnoj bazi.

Konfigurisanje Sinhronizacije

Sam proces sinhronizacije počinje pozivom metode Synchronize() od strane SyncAgent-a.

```
remoteSyncProvider.SyncProgress += new
EventHandler<SyncProgressEventArgs>( remoteSyncProvider _SyncProgress)
remoteSyncProvider.ApplyChangeFailed += new
EventHandler<ApplyChangeFailedEventArgs>(
remoteSyncProvider_ApplyChangeFailed)
localSyncProvider.ApplyChangeFailed += new
EventHandler<ApplyChangeFailedEventArgs>(
localSyncProvider_ApplyChangeFailed)
SyncStatistics syncStats = syncAgent.Synchronize()
localSyncProvider.UpdateSynchronizationVersion()
```

Ažuriranje verzije sinhronizacije

Definisanjem ove komande nakon sinhronizacije se ažurira tabela koja čuva verzije sinhronizacije u lokalnoj bazi.

```
SqlCommand command = new SqlCommand();
command.Connection = (SqlConnection)this.Connection;
command.CommandText = " UPDATE versionSync SET
newVersion = change_tracking_current_version() WHERE table = "+@tableName;
command.Parameters.Add("@tableName",SqlDbType.Nvarchar,50)
command.Connection.Open();
command.ExecuteNonQuery();
command.Connection.Close();
command.CommandText = "UPDATE VersionSync SET
LatestVersion = NewVersion WHERE table = "+@ tableName;
command.Parameters.Add("@tableName ",SqlDbType.Nvarchar,50)
command.Connection.Open();
command.ExecuteNonQuery();
command.Connection.Close();
```

5.5.4 Rezultati razvoja servisa za sinhronizaciju podataka

U sledećoj tabeli su dati primeri na osnovu kojih je predstavljena logika sinhronizacije koja je implementirana prilikom razvoja servisa za sinhronizaciju podataka.

Lokalna baza	Udaljena baza	Akcija pri sinhronizaciji
Unos novog sloga	Ne postoji slog	Unos novog sloga
Ne postoji	Unos novog sloga	Unos novog sloga
Obrisan slog	Bez promene sloga	Brisanje sloga
Bez promene sloga	Obrisan slog	Brisanje sloga
Obrisan slog	Ne postoji slog	Bez akcije
Ne postoji slog	Obrisan slog	Bez akcije
Obrisan slog	Obrisan slog	Bez akcije
Obrisan slog	Izmenjen slog	Na lokalnoj bazi se podatak ponovo unosi sa izmenom
Izmenjen slog	Obrisan slog	Na udaljenoj bazi se podatak ponovo unosi sa izmenom
Promenjen	Bez promene	Promenjen na udaljenoj bazi
Bez promene	Promenjen	Promenjen na lokalnoj bazi
Promenjen	Promenjen	Konflikt

Obzirom da aplikacija radi asinhrono, postoji mogućnost pojave konflikata prilikom usklađivanja baza.

Sama logika rešavanja konfliktnih situacija je smeštena u procedure, koje vrše unos, ažuriranje i brisanje podataka. Pre nego što se izvrši neka od ovih DML operacija, ispituje se da li će njenim izvršavanjem doći do konflikta među podacima. U slučaju da se detektuje potencijalni konflikt, primenjuje se unapred definisano pravilo za njegovo rešavanje, gde će jedna strana koja učestvuje u sinhronizaciji pobediti i nadjačati onu drugu. Pri tome se konfliktni slogovi unose u tabele konflikata koje postoje za svaku od tabela. Nakon izvršenja sinhronizacije, korisnik je obavešten ako je došlo do konflikta i u

mogućnosti je da pogleda slog koji je zabeležen u tabeli konflikata i da eventualno izmeni ishod arbitraže konfliktne situacije.

6 ZAKLJUČAK

Savremena preduzeća se često susreću sa problemom velike količine podataka, koja nastaje tokom poslovanja i sa još većim problemom održavanja raspoloživosti tih podataka. Alati za sinhronizaciju u distribuiranim sistemima baza podataka sa ponovljenim podacima imaju svoju primenu u rešavanju ovih problema, omogućavajući velikom broju aplikacija, koje rade na ovakvim sistemima, neprekidno funkcionisanje i raspoloživost korisnicima.

Svrha ovog rada je ovladavanje teorijskim konceptima distribuiranih baza podatka, spoznavanje kompleksnosti i poteškoća u funkcionisanju jednog takvog sistema, a sve u cilju primene tog znanja u konkretnom poslovnom okruženju. Problem sinhronizacije podataka treba shvatiti kao izazov kojem treba posvetiti pažnju i uzeti ga u obzir od početnih faza konceptualnog modeliranja programskog proizvoda, preko definisanja arhitekture i dizajna sistema sve do same implementacije. Danas su nam na raspolaganju razne tehnologije i alati za rešavanje ovih problema, ali je poznavanje teorijskog okvira nešto što treba da prethodi upuštanju u rešavanje konkretnog problema.

Konkretan primer razvoja servisa za sinhronizaciju podataka, uz korišćenje novih Microsoft tehnologija Synchronization Services i Change Tracker-a, ilustruje i praktično primenjuje teorijski koncept sinhronizacije u distribuiranom sistemu primenom lenje strategije distribuiranog ažuriranja. Realizovano praktično rešenje predstavlja samo osnov za dalji razvoj u realnom sistemu.

LITERATURA

1. M. Tamer Özsu, P. V. Principles of Distributed Database Systems, III izdanje, Springer, 2011.
2. Pavlovic-Lažetic, G. Uvod u relacione baze podataka, II izdanje, Matematički fakultet, 1999.
3. Lijun (June) Gu, LIoyd Budd, Aysegul Cayci, Colin Hendricks, Micks Purnell, Carol Rigdon, A Practical Guide to DB2UDB Data Replication V8 (http://www.redbooks.ibm.com/abstracts/sg246828.html)
4. http://www.oracle.com, januar 2011.
5. http://msdn.microsoft.com, januar 2011.
6. http://www.idc.com, jun 2011.
7. http://developer.apple.com, januar 2011.
8. http://www.xtivia.com, januar 2011.
9. http://www.mssqltips.com, novembar 2010.
10. http://blogs.msdn.com, januar 2011.
11. http://www.codeproject.com, januar 2011.
12. http://www.allthingsdistributed.com, januar 2011
13. http://www.sybase.com, decembar 2010
14. http://social.microsoft.com, decembar 2010
15. http://technet.microsoft.com, decembar 2010
16. http://www.blurtit.com, januar 2011
17. http://open-tube.com/nosql-for-dummies-rise-of-non-relational-database-engines/
18. http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php
19. http://www.royans.net/arch/brewers-cap-theorem-on-distributed-systems/