ZÁPADOČESKÁ UNIVERZITA
KATEDRA INFORMATIKY A VÝPOČETNÍ TECHNIKY

Semestrální práce z KIV/DS

Simulace transakcí

Martin Sloup

7.11.2010

Zadání

Případ užití, motivace

Databázový server obsluhuje paralelně několik klientů, kteří provádí v transakcích operace čtení/zápis nad daty ve sdíleném úložišti. Více klientů může zároveň v konkurentních transakcích přistupovat ke stejným položkám - server musí řešit přístupové konflikty.

Požadavky

- Implementace v C/C++.
- Překlad programem `make`.
- Žádné GUI, jen příkazový řádek, neinteraktivní - spustit, spočítat, zalogovat.
- OS Linux.
- Aplikace se sestává ze 2 hlavních spustitelných programů - server a klient.
- Paralelizace serveru volitelně vlákna nebo procesy (dále v textu jen vlákna).
- Množství klientů variabilní.
- Tabulka bude mít 100 řádků.
- Tabulka bude mít 3 sloupce:
  - identifikátor `id` - číslo řádku počítáno od 0, není tedy nutné, aby se implementovat,
  - hodnota typu `int` (celá čísla se znaménkem) - počáteční hodnota = `id` - 50 (tedy -50, 49, ..., 49),
  - časová značka posledního zápisu - počáteční hodnota 0, inkrementace po každém zápisu.
- Klient bude provádět operace podle zadaného dávkového souboru instrukcí.
- Druhy instrukcí:
  - `BEGIN` (začátek transakce)
  - `COMMIT` (spáchání transakce)
  - `ADD` *`src1_id src2_id dst_id`* (dst = src1 + src2, *_id jsou identifikátory řádků)
  - `SLEEP` *`ms`* (uspání klienta)
- Předpoklady:
  - Syntaxe a sémantika instrukční dávky bude vždy správná (žádné překlepy, spárované `BEGIN` a `COMMIT`, správné argumenty, ...).
  - Žádné vnořené subtransakce (transakce v transakci).
  - Instrukce `ADD` může být vykonána pouze v transakci (mezi instrukcemi `BEGIN` a `COMMIT`)
  - Instrukce `SLEEP` může být kdekoliv
  - V jedné transakci může být více instrukcí SLEEP i ADD.
- Komunikace se serverem probíhá jen při instrukcích `BEGIN` a `COMMIT`.
- Při instrukci `BEGIN` si klient vyžádá od serveru všechny hodnoty a jejich časové značky, které bude potřebovat v rámci této transakce.
- Při instrukci `COMMIT` klient odešle svá zpracovaná (změněná) data serveru společně s původními časovými značkami a získá od serveru odpověď, zda bylo spáchání úspěšně provedeno.
- Server bude řešit konflikty zápisů podle optimistického algoritmu s časovými značkami (RTS, WTS, TS).
- Vygenerujte jakýmkoliv způsobem datový soubor pro net_flow_vizu, kde budou zanesené datové přenosy, stavy lokálních kopií klientů na hranicích transakcí a konfliktní situace na serveru.
- Příjem a vysílání zpráv musí být vhodně paralelizované, neměly by se vzájemně blokovat.
- Dokumentace jen stručně v bodech, použité algoritmy, mechanizmy, ...

Řešení

Semestrální práce se skládá ze dvou samostatných aplikací. Z transakčního serveru a jeho klienta.

Transakční klient provádí zpracování předpřipraveného skriptu. Po spuštění se nejprve klient připojí na transakční server a načte skript, kde se nachází posloupnost příkazů, které jsou postupně vykonány. Klient podporuje příkazy dle zadání.

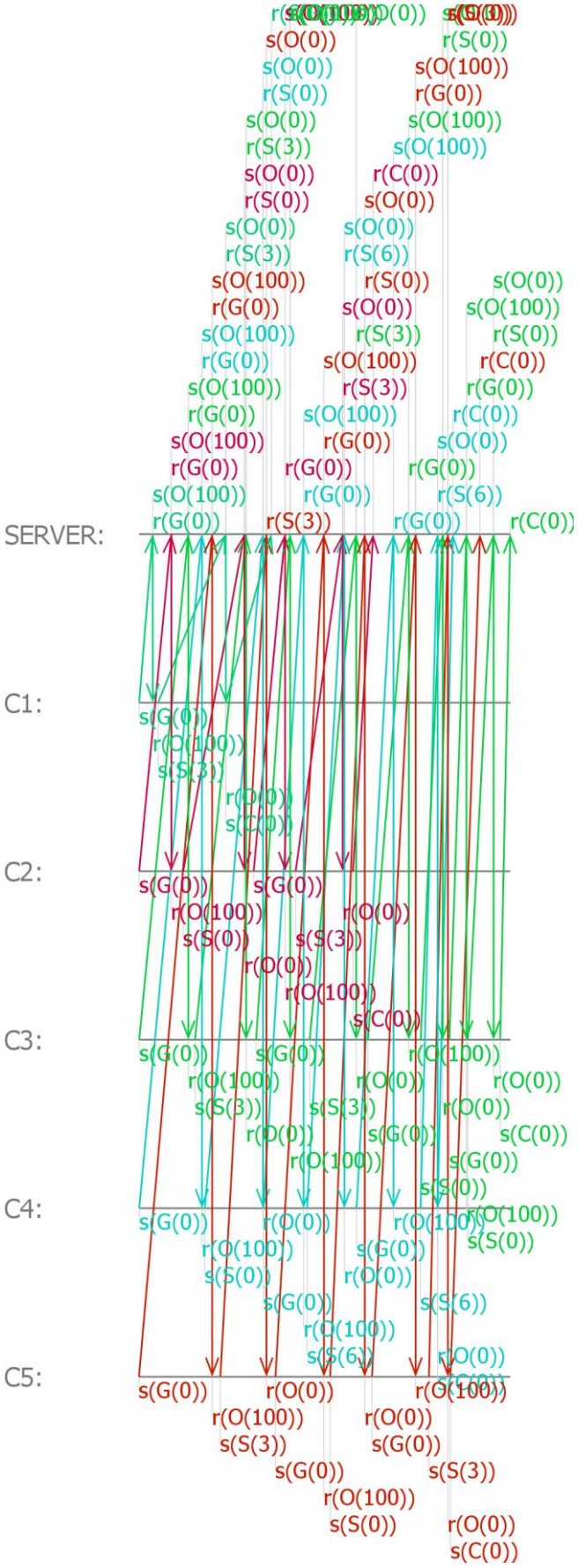
Příkaz `BEGIN` načte seznam proměnných z transakčního serveru včetně časového razítka. Toto časové razítko je zde důležité pro udržení konzistentního stavu při vykonání příkazu `COMMIT`, tedy odeslání seznamu použitých proměnných zpět na transakční server. Při odeslání je nutné zkontrolovat, zda již nějaký jiný klient neprovedl modifikaci proměnné. Pokud se tak stane, server při nahrání seznamu proměnných na server vrátí zpět klientovi chybu. Klient pak následně přeruší vykonávání skriptu, nebo pokud je klient spuštěn s parametrem „`-f`“, zobrazí se varování a pokračuje dál. Zadání zcela neříká, jak se má v této situaci klient rozhodnout. Proto byl přidán tento parametr. Mezi další příkazy patří příkaz `ADD`, který do proměnné zadané ve třetím parametru příkazu uloží součet proměnných zadaných přes první a druhý parametr. Nakonec příkaz `SLEEP` umožňuje pozastavení vykonávání skriptu na dobu zadanou prvním parametrem (v milisekundách). Po dokončení vykonávání skriptu se klient odpojí od serveru a ukončí se.

Transakční server je vcelku jednoduchý. Pouze obsluhuje požadavky od klientů a snaží se je atomicky uložit. Po spuštění nejprve načte výchozí hodnoty proměnných ze souboru a následně čeká na příchozí spojení od klientů. Spojení od klienta zůstává otevřené, dokud klient nepošle operaci `CLOSE`. Server implementuje dvě nejdůležitější operace, tj. `GET` a `SET`. Operace `GET` vrací všechny načtené proměnné klientovi, kdežto operace `SET` přijme seznam změněných proměnných a porovná je s databází, zda nedošlo k jejich změně a zda je možné změny uložit do databáze (kontrola časového razítka). Po uložení změny dojde ke zvýšení hodnoty časového razítka, což symbolizuje změnu proměnné, a pokud jiný klient při `COMMIT` má staré razítko u použitých proměnných, dojde k chybě.

Závěr

Naprogramovat algoritmus pro provádění transakcí nebylo složité. Původně bylo v plánu použít forkování po připojení klientů na transakčním serveru, ale to se ukázalo být problematické z důvodu sdílených struktur v paměti. Místo toho byla nakonec použita vlákna ve formě implementace přes knihovnu pthread. Zpracování instrukcí `SET` a `GET` na transakčním serveru není dokonalé. Aktuálně se používají mutexy na zachování atomicity a tím prakticky není možné vykonávat operaci `GET` pro více klientů zároveň.

Příloha (diagram komunikace)



Legenda:

G(0) – požadavek na stáhnutí seznamu všech proměnných (BEGIN)
S(počet) – odeslání změněných porměnných zpět na server (COMMIT)
O(počet) – odpověď serveru s udaným počtem položek
C(0) – uzavření spojení se serverem