Optimalizace plnění a aktualizace velkých tabulek

Milan Rafaj, IBM

Agenda

- OLTP vs DSS zpracování
- Optimalizace INSERT operací
- Optimalizace DELETE operací
- Optimalizace UPDATE operací
- Zdroje
- Dotazy

OLTP vs DSS zpracování

- OLTP
  - Silně normalizovaný datový model
  - Rozsáhlé využití indexů a referenční integrity
  - Transakční zpracování
  - Transakce aktualizují většinou malé množství dat
- DSS
  - Obvykle denormalizovaný datový model
  - Omezené použití indexů
  - Dávkové zpracování
  - Aktualizace velkého množství dat

OLTP vs DSS zpracování

- Použití OLTP technik aktualizace dat při aktualizaci v DSS systémech vede k neúnosným časům zpracování a enormnímu počtu I/O operací
- Příklad: operace insert do tabulky s primárním indexem bránícím duplicitám si vyžaduje následující operace:
  - vyhledání v indexu (u velkých tabulek až 6 I/O operací)
  - pokud se v indexu nanašel, vloží se řádek na stránku s volným místem (minimálně 1 I/O read operace)
  - potenciální alokace nového extentu v tabulce
  - vložení nové hodnoty do indexu
  - potenciální split indexového stromu
  - potenciální alokace extentu indexu
  - „before image“ je zapsán do fyzického logu
  - operace INSERT je zapsáno do logického logu
  - periodicky se stránka zapisuje na disk (checkpoint, asynchronní nebo foreground write)

Představme si tento proces při vkládání několika miliónů řádků!
Představme si další režii, pokud tato dávka běží jako transakce a dojde k pokusu o vložení duplicitního řádku!
Další režii je spojená s přenosem stránek mezi diskem a buffer cache pamětí!
Velká tabulka a její indexy také degradují kvalitu buffer cache paměti

Optimalizace operací INSERT

- Problém
  - Z předchozího vyplývá následující:
    - Operace insert procházejí přes buffer cache
    - Pro každý vkládaný řádek se prohledává index
    - Duplicitní indexová hodnoty znemožní provést celou dávku jako jednu transakci
      - vede na kurzorové zpracování a velké množství samostatných transakcí
      - částečně řešitelné pomocí violation tables a nastavením módu FILTER

Eliminace použití buffer cache

- IDS umožňuje číst i zapisovat data bez využití buffer cache paměti
- Light scan – operace čtení
  - Řádky se čtou do privátních oblastí ve virtuální paměti
  - Buffer cache není degenerována daty z rozsáhlých tabulek
  - Podmínky light scan
    - Úroveň izolace dirty read nebo raw table
    - Proměnná prostředí LIGHT_SCANS=FORCE
    - Velikost tabulky 1MB (>buffer cache IDS 10.x-)
    - onconfig parametry RA_PAGES a RA_TRESHOLD
    - onconfig parameter BATCHEDREAD_TABLE 1

Eliminace použití buffer cache

- Light Append – operace zápis
  - Podmínky
    - Tabulka není žurnálovaná a je dočasná
    - Databáze není žurnálovaná a tabulka se plní pomocí HPL nebo z externí tabulky v režimu express
    - Tabulka je typu raw a plní se pomocí HPL nebo z externí tabulky v režimu express
    - XPS – tabulka je typu raw a plní se operací insert ... select
  - Řádky se vkládají na nové stránky v privátní oblasti virtuální paměti a ty se po dokončení operace připojí na konec tabulky
  - onconfig parametry RA_PAGES a RA_TRESHOLD

Eliminace vyhledávání v indexu

- Cíl – vkládat jen neduplicitní řádky
- Metoda – využití operace hash-join a pomocné tabulky

Paralelismus a fragmentace

- Abychom mohli operaci INSERT ještě více zefektivnit, je velmi důležité rozsáhlé tabulky fragmentovat a při operacích v maximální míře použít PDQ

Optimalizace operace INSERT - řešení

1. Zrušení indexu na cílové tabulce
2. Případně změníme typ cílové tabulky na raw
3. Vytvoříme raw table pro nová data a naplníme ji (od verze 11.50.FC6 lze využít externí tabulky)
4. Provedeme update statistics low pro raw table (light scan)
5. Vytvoříme dočasnou tabulku s duplicitními řádky/klíči (light scan,hash-join a light append)

```
select r.klic from cilova c,raw r where c.klic=r.klic
into temp unik with no log
```

   - můžeme dočasnou tabulku fragmentovat
   - Pokud je prázdná, můžeme jednoduše přidat raw do cílové tabulky
6. Vložíme do ní i všechny klíče vstupní tabulky (light scan a light append)

```
insert into unik select klic from raw
```

Optimalizace operace INSERT - řešení

7. Vytvoříme dočasnou tabulku s neduplicitními klíči (light scan a light append)
```
select klic from unik group by klic
having count(*)=1 into temp unik_klic with no
log
```
8. Vložíme do cílové tabulky neduplicitní data (light scan, hash-join, parallel insert)
```
insert into cil select r.* from raw r,unik_klic
where unik_klic.klic=r.klic
```
9. Změníme typ cílové tabulky na standard
10. Znovu vytvoříme index cílové tabulky

Testovací případ

- Power6 1CPU, DS_TOTAL_MEMORY 800000
- Tabulka cil(i integer, v integer, a char(104))
  - 55 000 000 řádků, cca 8GB
  - Fragmentovaná round robin do 2 dbspace
- Tabulka raw(i integer, v integer, a char(104))
  - 4 000 000 řádků, cca 512MB
  - 50% (2 000 000 řádků) duplicit tj. 3.6% tabulky cil

Tabulka výsledků testů

Počáteční naplnění cil (buffer)	8m19s
Počáteční naplnění cil (LAP)	2m6s
Počáteční naplnění raw (LAP)	16s
Insert neduplicitních řádků	3m8s
Hash join cil a raw paralelní	1m45s
Řešení duplicit	20s
Insert neduplicitních (raw bez duplicit)	7s
Insert neduplicitnich (hash join raw a unik)	40s
Update duplicit	Zmíněno později

Shrnutí

- Eliminovali jsme použití indexů
- Eliminovali jsme maximálně použití buffer cache
- Potenciální problémy
  - Dostatek CPU a RAM pro PDQ
  - Velikost dočasných tablespace (probe tabulka operace hash join je poměrně náročná na paměť)

Obecná doporučení

Nedělejte:	**Dělejte:**
Neuvažujte v pojmech řádků	Pracujte se všemi řádky najednou
Nepoužívejte logování transakcí	Eliminujte maximum indexů
Nepoužívejte referenční integritu	Kombinujte data do jediné tabulky, operace join je drahá, pokud jsou data v jedné tabulce, práce bude mnohem snadnější
Malé tabulky (maximum 255 řádků na stránce)	Rozumně velké tabulky (délka řádku min. 100b) – scan tabulky 12b 50MB/s scan tabulky 112 b 150MB/s !!!
Nepoužívejte kurzory – zbavujete se výkonu databázového stroje	Extrahujte pracovní skupinu řádků, zpracujte je a vložte zpět do nové kopie tabulky, do které jste přidali nezměněnou skupinu řádků

Optimalizace operace UPDATE

- Metodologie aktualizací ze světa OLTP je nepoužitelná a vedla by k mnoha hodinovému zpracování
- Příklad

```
update cil set cil.v =
    (select r.v from raw r
     where r.klic = c.klic)
where cil.klic in
    (select klic from raw)
```

předpoklad, že v raw jsou jen aktualizované řádky

Varianta 1 – UPDATE JOIN

IDS 11.50 umožňuje operace MERGE:

```
merge into cil using raw as r on
cil.klic=r.klic
when matched then update set (cil.v,cil.a) =
(r.v,r.a)
when not matched then insert
values(r.klic,r.v,r.a);
```

- Operace použije hash-join, je velmi náročná na dočasný prostor a je poměrně pomalá☹

Varianta 2 – Aditivní UPDATE

1. Vytvoření nové cílové raw tabulky
2. Vytvoření a naplnění raw tabulky se změnovými daty
3. Naplnění nové cílové tabulky aditivní aktualizací hodnot

```
insert into novy_cil
select klic,nvl(cil.sl1+raw.sl1,cil.sl1),...
from cil,outer raw
where cil.klic=raw.klic --  union pokud sl1 má -
-- null hodnoty
```

4. Zrušení původní cílové tabulky
5. Přejmenování nové cílové tabulky a změna na standard
6. Vytvoření indexů nad cílovou tabulkou

Varianta 2 – klasický UPDATE

1. Vytvoření nové cílové raw tabulky
2. Vytvoření a naplnění raw tabulky se změnovými daty
3. Naplnění nové cílové tabulky klasickou aktualizací hodnot

```
insert into novy_cil
select klic,nvl(raw.sl1,cil.sl1),...
from cil,outer raw
where cil.klic=raw.klic
```

4. Zrušení původní cílové tabulky
5. Přejmenování nové cílové tabulky a změna na standard
6. Vytvoření indexů nad cílovou tabulkou

Varianta 3 – klasický upsert

1. Vytvoření nové cílové tabulky
2. Vytvoření a naplnění raw tabulky se změnovými daty
3. Vložení všech hodnot změnové tabulky
   ```
   insert into novy_cil select * from raw
   ```
4. Vložení neaktualizovaných hodnot původní tabulky
   ```
   insert into novy_cil from cil
   where i not in (select i from raw)
   ```
5. Zrušení původní cílové tabulky
6. Přejmenování nové cílové tabulky a změna na standard
7. Vytvoření indexů nad cílovou tabulkou

Tabulka výsledků testů

Počáteční naplnění raw	16s
Aditivní update	13m3s
Klasický update	11m30s
Upsert (spojení insert a update metody)	12m38s
MERGE (autoindex cil)	53m+ (plný disk)
MERGE (hash join)	53m+ (plný disk)

Shrnutí

- Eliminovali jsme maximálně buffer cache
- V případě aditivního a klasického UPDATE do nové cílové tabulky se využije pouze light scan a insert, nemusí se přepisovat stránky (další režie) jako u varianty s příkazem MERGE
- Předpokladem je velký dostatečně velký počet aktualizovaných či nových řádků

Optimalizace operace DELETE

- Výmaz velkého počtu (statisíce až milióny) řádků z obrovské tabulky (desítky milionů řádků) pomocí indexu může trvat hodiny
- Operace DELETE si vynucuje
  - Přepsání stránky
  - Vznik prázdných míst na stránkách, které se většinou již v DSS aplikacích nemusejí znovu zaplnit

Varianta 1 – delete tabulky

- NOT IN subquery (light scan a hash-join)

```
delete from cil
where klic in
   (select klic from rukl)
```

- NOT EXISTS subquery (light scan a nested scan)

```
delete from cil
where exists (select 0 from rukl
                  where
rukl.klic=cil.klic)
```

Varianta 2 – přepis tabulky

- Vytvoříme novou cílovou tabulku pouze s řádky, které chceme zachovat (light scan a hash-join)
- NOT IN subquery

```
insert into novy_cil
select * from cil
where klic not in
    (select klic from rukl)
```

- NOT EXISTS subquery (light scan a nested scan)

```
insert into novy_cil
select * from cil
where not exists (select 0 from rukl
                  where rukl.klic=cil.klic)
```

Tabulka výsledků testů

Počáteční naplnění raw	16s
Delete + not in subquery	9m36s
Delete + not exists subquery	10d!!!
Insert + not in subquery	12m9s*
Insert + not exists subquery	10d!!!
*insert trvá déle, ale s rostoucím počtem mazaných řádků se poměr otočí	

Shrnutí

- Varianta přepisu tabulky je efektivnější s rostoucím počtem řádků, které je třeba vymazat, protože doba hash-join zůstává konstatní, ale ubývá zápisů do nové cílové tabulky
- Předpokladem je větší počet řádků (tisíce, milióny), jinak je klasický delete přes index samozřejmě efektivnější

Zdroje

- Jack Parker – Decision Support System (DSS) Application Processing
  http://www.ibm.com/developerworks/data/zones/informix/library/techarticle/parker/0502parker.html
- Informix SQL Syntax

Dotazy