De Logische I/O

Kosten gemakkelijk te beïnvloeden

In de Oracle database speelt de logische I/O een belangrijke rol, maar nergens wordt echt goed beschreven wat een logische I/O is. Ook lijkt het er in eerste instantie op, dat elke logische I/O hetzelfde is wat betreft de kosten. Toch is er een groot verschil tussen de logische I/Os. Dit artikel gaat wat dieper in over de LIO (logisch I/O) in Oracle.

Oracle beschikt over een buffer cache die blokken uit de database files cached. In deze blokken zitten één of meer rijen (rows). Het lezen van het blok uit de file is een zogenaamde Physieke I/O. Een PIO betekent niet dat de data van de harde schijf hoeft te komen. Het kan namelijk ook al gecached zijn in een Disk Array of de buffer cache van een Operating System. Dus de PIO betekent dat Oracle het Operating Systeem vraagt een blok te lezen.

In Oracle 9i is een interessante uitbreiding doorgevoerd in v$sqlarea

LIO en PIO

Wanneer dit blok in de Oracle buffer cache is gelezen kan het benaderd worden om één of meer rijen te lezen of te veranderen in dat blok. Het kan ook zijn dat je een blok moet benaderen om te zien dat geen enkele rij in het blok voldoet aan een bepaald zoekcriterium. Toch moeten dan alle rijen benaderd worden om te kijken of een rij voldoet aan een zoek criterium. Dus er zijn al verschillende LIOs te onderscheiden:

- Een LIO die een of meer rijen terug brengt.
- Een LIO die geen rijen terug brengt voor een zoek criteria.
- Een LIO die geen rijen terug brengt omdat de rijen verwijderd worden.

Oracle heeft verschillende statistieken voor deze LIOs die terug te vinden zijn in v$sysstat:

- Consistent gets
- Db block gets
- Consistent changes
- Db block changes

Voor de PIO heeft Oracle ook een aantal statistieken in v$systat maar ook in v$system_event:

- Physical reads

- Db file sequential read
- Db file scattered read

Een belangrijk weetje hier is dat physical reads aangeeft hoeveel blokken er gelezen zijn en dat db file sequential read en db file scattered read aangeven hoeveel keer er gelezen is. Oracle kan namelijk een enkel blok lezen (db file sequential read) en meerdere blokken (db file scattered read).

Buffer Cache Hit ratio

De buffer cache hit ratio is één van de meest bekende hit ratio's die in Oracle Performance Monitoring en Tuning wordt gebruikt. De hit berekent de ratio tussen hoe vaak de Oracle buffer cache is benaderd en hoe vaak Oracle een block uit een file heeft moeten lezen. De ratio wordt dus als volgt berekend:

$$\frac{LIO - PIO}{LIO} * 100\%$$

Uit tabel 1 blijkt wel dat er veel meningen zijn over een goede BCH, zelfs Oracle geeft geen eenduidige opvatting. Alle bronnen gaan ervan uit dat het berekenen van de BCH zinvol is. Maar is dat zo?

BCH	Source
> 80% (UNIX files) or > 90% (raw devices)	Oracle
> 95% online en > 85% batch	Gurry & Corrigan
94 –97% in Oracle Applications	Oracle
99.99999%	Niemitz
94 – 97% in Oracle Applications	Oracle
> 80% (UNIX files) or > 90% (raw devices)	Oracle
> 90%	Don Burleson

Tabel 1. Verschillende bronnen hanteren verschillende normen voor een goede BCH

Kosten van een LIO

Binnen Oracle wordt een LIO berekent als "consistent gets" + "db block gets". Dit geeft weer het aantal operaties en niet hoe duur ze waren (geen kosten). Kijk je bijvoorbeeld in v$filestat dan zie je het aantal lees operaties maar ook hoe duur ze waren (PHYRDS = aantal en PHYRDTIM = kosten). Zoals men weet zal niet elke PIO even duur zijn (dat hangt bijvoorbeeld af van welke schijf men benadert en of er een cache is). Een ander probleem is dat men niet kijkt of de LIO wel nodig was, sterker nog: de formule geeft een beter resultaat (dichter tegen de honderd procent) als de LIO erg hoog is (of PIO erg laag is).

Een belangrijk weetje is dat physical reads aangeeft hoeveel blokken er gelezen zijn

Hoe kunnen we nu de kosten bekijken van een LIO? Oracle9i heeft een uitbreiding in v$sqlarea die erg interessant is. Er zijn twee kolommen toegevoegd, namelijk CPU_TIME en ELAPSED_TIME. Beide worden in microseconden bijgehouden voor een SQL statement. Daarnaast worden ook de LIO en de PIO per SQL statement bijgehouden (BUFFER_GETS en DISK_READS). Dus door de CPU_TIME te delen door het aantal BUFFER_GETS, krijg je de kosten voor een LIO.

De volgende tests kunnen we doen in de volgende omgeving:

```
Create table testing as select * from sys.obj$;
Select count(*) from testing
31737 rows (op het test systeem)
```

```
Select sql_text, cpu_time, elapsed_time, buffer_gets
From v$sqlarea
```

[1]

```
SQL_TEXT                          CPU_TIME   ELAPSED_TIME   BUFFER_GETS
-------------------------------------------------------------------------
select /* 400 */ obj# from testing    80000          70110       472
select /* 1 */ obj# from testing    1490000        1118524       16055
```

Beide SQL statements selecteren alle rijen van de testing table. Alleen de eerste test heeft een array size van 400 en de tweede test een array size van 1 en dit gebeurt gewoon via SQL*Plus. Het blijkt dus dat een array fetch van 400 er voor zorgt dat alles bijna twee keer sneller draait.

De LIO kost in beide gevallen:

a. 80000/472 = 169.49 micro seconde
b.1490000/16055 = 92.81 micro seconde

Dus de eerste test is sneller, maar de LIO is bijna twee keer zo duur. Het verschil zit in het aantal LIO dat gedaan is in beide tests (ruim 400 tegen 16000). Het grappige is dat de BCH voor het eerste statement beduidend slechter is dan het tweede statement, terwijl het eerste statement veel sneller is.

Het zal u zijn opgevallen dat de CPU_TIME groter is dan de ELAPSED_TIME. Dit is geen fout in de data, maar een probleem dat ontstaan is doordat de CPU-tijd eigenlijk in centiseconden wordt gemeten en later naar microseconden wordt geconverteerd. De elapse time wordt netjes bijgehouden in microseconden.

De CPU-tijd in dit geval is eigenlijk alles wat je nodig hebt om een SQL statement uit te voeren, zoals het parsen, de LIO, het vinden van de rijen in de blokken en het terugsturen van de data naar de client.

Wat zou er gebeuren als een SQL toch alle rijen moet bekijken maar niks terug stuurt naar de client? Daarvoor doen we de volgende test:

```
Select /* < */ obj# from testing where obj# < 0
```

[1] Waarom is het verschil zo groot tussen CPU_TIME en ELAPSED_TIME voor select / 400 / en select /* 1 */ ?*

CPU_TIME	ELAPSED_TIME	BUFFER_GETS
	30000 33772	393

30000/393 = 76.34 microsecond per logical I/O.

De kosten bedragen nu 76.34 micro seconden in plaats van 169.49 en 92.81. Het verschil kan dus worden gezien als het oversturen van de data.
De obj# kolom is steeds de eerste kolom van de tabel. Wat zou er gebeuren als we de laatste kolom in de query betrekken?

```
Select /* < */ obj# from testing where spare6 is not null
```

CPU_TIME	ELAPSED_TIME	BUFFER_GETS
50000	45570	393

50000/393 = 127.23 microsecond per logical I/O.

Hieruit blijkt dat de positie van de kolom die wordt gebruikt in de query mede de kosten van de LIO bepaalt. Het betekent dus dat de Oracle kernel door alle kolommen van de tabel moet scannen om bij de laatste kolom te komen (sequentiële access), er is geen directe access naar de laatste kolom mogelijk.

De kosten van een LIO zijn gemakkelijk te beïnvloeden met een grote array fetch size

Een interessante conclusie die men hieraan kan verbinden is dat de meeste benaderde kolommen van een query vooraan in de tabel moeten staan, zodat de LIO het goedkoopst wordt. Door nu nog een keer de tweede kolom van de tabel te pakken, zien we dat het klopt.

```
Select /* < */ obj# from testing where dataobj# < 0
```

CPU_TIME	ELAPSED_TIME	BUFFER_GETS
30000	30250	393

30000/393 = 76.34 microsecond per logical I/O.

Deze kosten zijn gelijk aan de kosten van de query die in de where clauseobj# < 0 had staan.

Conclusie

Er wordt vaak naar het aantal LIO's in Oracle gekeken, maar zelden naar de kosten ervan. Het blijkt dat de kosten van een LIO gemakkelijk zijn te beïnvloeden door te werken met een grote array fetch size of door de positie van de kolommen te veranderen zodat de meest benaderde kolommen vooraan in de tabel komen te staan. De Buffer Cache Hitratio kijkt alleen maar naar aantallen van LIO, niet naar de kosten. Door de kosten van de LIO te beïnvloeden kan de query en performance van een systeem positief veranderen.

Anjo Kolk
is chief Oracle technologist bij Precise Software Solution. Voordat hij bij Precise Software Solutions in dienst trad, werkte hij meer dan zestien jaar bij Oracle Corporation, waar hij zich vooral bezighield met rdbms-performance. Anjo schreef het bekende whitepaper 'Oracle7 wait events and enqueues', moderator van de website http://www.oraperf.com en geestelijk vader van YAPP methode (op responstijden gebaseerde fine tuning van Oracle systemen). Hij is per e-mail te bereiken op akolk@precise.com.