Doe-het-zelf-sessie
Oracle performance-onderzoek en optimalisatie

ciber
ALWAYS ABLE

Rob van Wijk

20 maart 2008 en 9 april 2008

Inleiding

Iedereen die met databases werkt, komt met enige regelmaat in aanraking met performance-problemen: de ontwikkelaar die ziet dat zijn zoekvraag, die in een fractie van een seconde resultaten had moeten opleveren, er meerdere seconden over doet, de DBA die opeens van gebruikers telefoontjes krijgt dat de database die hij beheert niet meer vooruit te branden is en de BI-er wiens management-rapportages niet meer voor 7 uur 's ochtends tot een einde gekomen zijn. Vaak is de oorzaak van het probleem – achteraf gezien – heel logisch. De hamvraag zou moeten zijn hoe je zo effectief mogelijk die echte oorzaak kunt achterhalen. In de praktijk zie je echter vaak dat er niets onderzocht wordt, maar dat er maar wat wordt geprobeerd: een ontwikkelaar die lukraak zijn zoekvraag gaat herschrijven naar een functioneel equivalente zoekvraag, een DBA die wat database parameters gaat aanpassen of de BI-er die zijn zoekvragen van hints gaat voorzien. En dit alles slechts in de hoop dat het wat oplevert. Meestal leidt dat echter tot niets. Heel af en toe heeft iemand geluk wat dan weer leidt tot uitspraken als "IN is altijd sneller dan EXISTS", of "CURSOR_SHARING moet je altijd op FORCE zetten" of "gebruik altijd de PARALLEL-hint", welke dan weer enige tijd als mythe door Oracle-land blijven rondzweven.

Feit is dat de Oracle-database een goed geinstrumenteerd systeem is dat ons veel mogelijkheden geeft om te kunnen onderzoeken waar de tijd wordt gespendeerd. Met kennis van deze meetmethodes kan zo'n beetje alle informatie worden opgevraagd die je je maar wenst. Het enige dat dan nog kan gebeuren, is dat je niet weet hoe je deze informatie moet interpreteren. Vaak drijft echter de reden en/of de oorzaak meteen naar boven.

Het zwaartepunt van deze sessie ligt daarom bij de diverse meetmethodes, en daarnaast zal aandacht besteed worden aan belangrijke onderwerpen die bijdragen aan het begrijpen van de oorzaak van veel performance-problemen, waaronder bindingsvariabelen en peeking, histogrammen, partities. Uiteraard zullen we diverse performance-problemen niet alleen onderzoeken, maar ook daadwerkelijk oplossen.

De onderwerpen zijn:
1) De twee categorieën performance-problemen
2) SQL*Trace en tkprof
3) Explain plan
4) Statspack
5) Statistieken
6) Optimaliseren
7) Histogrammen
8) Bindingsvariabelen en peeking
9) PL/SQL bulkverwerking
10) Gepartitioneerde tabellen

Het belangrijkste doel dat vandaag geprobeerd wordt te bereiken, is dat je na afloop ervan overtuigd bent dat er altijd eerst onderzocht moet worden waar het echte probleem zit, om vervolgens het probleem met één gerichte actie op te lossen natuurlijk.

Voorbereiding

Download de Oracle-database versie 11.1.0.6 vanaf http://www.oracle.com/technology/software/products/database/index.html en installeer de database.

Tip: de Oracle-processen worden standaard opgestart als je laptop opstart. Dit kan je veranderen door in de Windows "Control Panel" → Administrative Tools → Services deze services op "Manual" te zetten.

Maak een startora11.bat met de inhoud:

```
net start <naam listener-service>
net start <naam database-service>
```

Op dezelfde manier maak je ook een stopora11.bat

```
net stop <naam listener-service>
net stop <naam database-service>
```

Deze sessie is ontwikkeld om te werken met SQL*Plus. In versie 11 is er geen Windows-versie meer, dus ik heb het hier over de "DOS-versie" sqlplus.exe. De opgaven en antwoorden zullen hierop aansluiten. Eigen favoriete GUI-toepassingen kan je ook gebruiken, mits ze SQL*Plus-scripts kunnen uitvoeren, maar zullen niet aansluiten. Daarom zou ik jullie willen aanraden om – voor eens in je leven ? – eens met SQL*Plus te werken.

Om SQL*Plus een beetje fraai te configureren, kan ik jullie het volgende artikel aanbevelen: http://www.williamrobertson.net/documents/sqlplus-setup.html.

Maak een gebruiker aan met jouw eigen naam of eigen code (bijvoorbeeld rwijk) en ken de DBA rol aan deze gebruiker toe. Installeer de bekende EMP en DEPT tabellen (en nog een paar andere tabellen) door het script demobld.sql te draaien. Dit script kan je vinden op internet door op "demobld.sql" te google-en.

Voer tenslotte onder de gebruiker SYS het volgende commando uit:

```
grant execute on dbms_alert to <jouw gebruikersnaam>;
```

1. De twee categorieën performance-problemen

Een actieve sessie is altijd hetzij werk aan het doen (verbruiken van cpu) hetzij aan het wachten. De doorlooptijd is de optelsom van de twee. Een performance-probleem kan je daarom indelen in één van de volgende twee categorieën:
1. de sessie doet voornamelijk niets omdat het ergens op aan het wachten is
2. de sessie doet voornamelijk teveel vanwege één of andere inefficiëntie

In geval 1 kan de sessie bijvoorbeeld wachten omdat het een rij probeert te bewerken die al door een andere sessie is vergrendeld, of omdat meerdere sessies op een latch zitten te wachten. In geval 2 zie je dat een zoekvraag meer systeembronnen gebruikt dan nodig. Het heeft meestal weinig zin je SQL te gaan optimaliseren in geval 1.

Een goede eerste stap voor het onderzoeken en uiteindelijk oplossen van een performance-probleem is daarom om de categorie te bepalen. Als systeembeheerder kan je bijvoorbeeld het UNIX-commando "top" gebruiken. Maar ook de v$-views bevatten genoeg informatie om dit te bepalen.

a) Lees in de Oracle Database Reference over de views V$SESSION, V$SESSION_WAIT, V$SESSION_LONGOPS en V$SESS_IO en schrijf een scriptje dat alle actieve sessies toont en voldoende informatie om de categorie te kunnen bepalen. Toon in ieder geval de module column van v$session. Geinstrumenteerde systemen zetten daar namelijk informatie in over de module die momenteel draait.

b) Start een losse SQL*Plus sessie op en start script ***opoo1b_installeer***. Bekijk in een andere sessie met behulp van het in 1a gemaakte script met welke categorie je te maken hebt. Als je klaar bent, start je script ***opoo1b_einde*** in de andere sessie om een en ander netjes te beëindigen.

c) Start een losse SQL*Plus sessie op en start script ***opoo1c_installeer***. Bekijk in een andere sessie met behulp van het in 1a gemaakte script met welke categorie je te maken hebt. Als je klaar bent, start je script ***opoo1c_einde*** in de andere sessie om een en ander netjes te beëindigen.

d) Bekijk eens vluchtig waarop je allemaal kunt wachten, door de volgende zoekvraag uit te voeren: ***select name from v$event_name order by name***

Extra) Pas het script uit opgave 1a dusdanig aan dat je nog meer nuttige zaken te weten kunt komen over drukke en/of wachtende sessies. Bijvoorbeeld: het SQL-commando dat momenteel uitgevoerd wordt en het object waarop wordt gewacht, als het wacht.

2. statspack

Statspack is het gereedschap voor de DBA die wil kunnen zien wat er allemaal gebeurt op zijn gehele database qua performance, over alle sessies heen.

Vanaf versie 10 geldt Automatic Workload Repository (AWR) als de opvolger van statspack. Echter is dit een extra te licenseren optie en statspack werkt in 11 nog

gewoon en geeft de meeste mensen waarschijnlijk al meer informatie dan je ooit zou wensen.

a) Log aan als SYS AS SYSDBA en draai het script dat statspack installeert: `@%ORACLE_HOME%\rdbms\admin\spcreate`. Geef als wachtwoord perfstat mee en laat de twee tabelruimtes leeg. Aan het einde ben je verbonden als perfstat.

b) Onder de gebruiker perfstat maak je een eerste snapshot door de procedure statspack.snap uit te voeren. Start vervolgens een tweede sessie op en voer hier minimaal 5 zoekvragen uit. Maakt niet uit wat. Als je hiermee klaar bent, maak je een tweede snapshot door weer statspack.snap uit te voeren.

c) Onder de gebruiker perfstat maak je nu het statspack rapport aan door een script te draaien: `@%ORACLE_HOME%\rdbms\admin\spreport`. Geef de twee statspack snap id's mee (de eerste keer is dit 1 en 2). Het rapport toont wat er tussen deze twee snapshots allemaal gebeurd is. Neem het rapport vluchtig door om een idee te krijgen welke informatie hier allemaal te vinden is.

Extra) Een AWR-rapport in HTML kun je krijgen door te draaien: `@%ORACLE_HOME%\rdbms\admin\awrrpt`. Kies dezelfde snap_id's als hierboven en kies html. Bekijk de gegenereerde HTML-pagina.

3. SQL*Trace en tkprof

Als eenmaal is bepaald dat het probleem in categorie 2 valt, oftewel er wordt (te) veel werk uitgevoerd, dan is het zaak om te bepalen waar de tijd precies gespendeerd wordt. Dit kan het allerbeste gedaan worden door een langzaam proces of zoekvraag te "tracen". Je zet dan SQL*Trace, zijnde Oracle's instrumentatie, aan. Zolang SQL*Trace is geactiveerd, schrijft Oracle veelvuldig diverse performance-gerelateerde informatie weg in een .trc bestand, in de folder die gespecificeerd staat in de USER_DUMP_DEST parameter. Het trace-bestand zelf is voor een ongeoefend oog nauwelijks leesbaar, en daarom is er tkprof. Tkprof is een programma buiten de database, dat een trace-bestand omzet in een leesbaarder tekstbestand.

a) bekijk waar de trace-bestanden terecht zullen komen op jouw laptop. Type hiertoe in: ***show parameter user_dump_dest***

b) Start een tweede SQL*Plus sessie op en start het script ***opoo3_installeer***, zonder te kijken wat dit doet.

c) Zet tracing in je eerste sessie aan door in te typen: ***alter session set sql_trace true***. Start de procedure opoo3 door in te typen ***exec opoo3*** en sluit alle cursoren door je sessie te beëindigen. Type daarom in: ***disconnect*** en log daarna weer aan. Als je wilt, kun je zien dat er door de laatste stap rijen in het trace-bestand zijn geschreven die beginnen met #STAT. Deze rijen bevatten de informatie die leidt tot de sectie "Row Source Operation" in het uiteindelijke tkprof-bestand.

d) Navigeer naar de USER_DUMP_DEST folder die je hebt bekeken in vraag 3a. Bekijk eerst eens alle opties voor tkprof, door alleen tkprof in te typen. Bekijk vervolgens het trace-bestand. Type daarna in:

tkprof <trace-bestand> a.txt sys=no sort=prsela exeela fchela

En bekijk vervolgens het bestand a.txt.

e) Alle informatie in het tkprof-bestand duidt precies wat er daadwerkelijk is gebeurd. Waar in dit proces werd de meeste tijd gespendeerd?

f) Lees paragraaf 21.4.4 "Interpreting TKPROF output" uit de Oracle Database Performance Tuning Guide t/m 21.4.4.6 "Library Cache Misses in TKPROF" (21-15 tot 21-18) en probeer alle getallen in het tkprof-bestand te begrijpen. Schrijf de secties/getallen op die niet helemaal duidelijk zijn.

Extra) Maak een kopie van procedure opoo3 door de broncode op te halen uit de user_source view, noem de nieuwe procedure opoo3_kopie en breng hierin een versnelling aan. Toon dit aan door een nieuw tkprof-bestand te fabriceren van de aangepaste procedure.

Extra 2) Experimenteer met het commando ***alter session set events '10046 trace name context forever, level N'***. Als N=0, dan is dit equivalent met ***alter session set sql_trace false***. Als N=1, dan is dit equivalent met ***alter session set sql_trace true***. Vul zelf de waarden 4, 8 en 12 in en bekijk wat de gevolgen zijn in je trace-bestand en het tkprof-bestand.

4. Explain plan

Een methode om alleen het plan op te vragen van een SQL-commando, is via het explain plan commando. Nadeel hiervan is dat je niet zeker bent dat het plan dat je ziet, ook het plan is dat gebruikt zal worden in de applicatie. Dit komt omdat omgevingsvariabelen, optimizer-instellingen, bindingsvariabelen anders kunnen zijn in de applicatie. Toch zijn er twee belangrijke voordelen:

- de zoekvraag zelf hoeft niet uitgevoerd te worden: alleen de parse-fase hoeft doorlopen te worden, en
- je krijgt te zien welk predicaat er in welke stap wordt toegepast

a) Lees in de Oracle Database Performance Tuning Guide hoofdstuk 12 tot en met paragraaf 12.5 (bladzijden 12-1 tot 12-8, dus stoppen bij de paragraaf "Viewing Parallel Execution with EXPLAIN PLAN").

b) Vraag het plan op van het SQL-commando uit script opoo4b:

```
select /*+ use_nl(s e) use_nl(e d) */
       e.ename
     , d.dname
     , s.grade
  from emp e
     , dept d
     , salgrade s
 where e.deptno = d.deptno
   and e.sal between s.losal and s.hisal
   and d.dname like '%S%'
 order by d.dname
     , e.ename
```

c) Een plan wordt altijd van binnenuit gelezen en bij stappen van gelijke diepte wordt de eerst genoemde eerder uitgevoerd. Het gemakkelijkst is dit te zien door de stappen in een boomstructuur te plaatsen. Teken een boom voor het plan uit

de vorige vraag en negeer ID 0 (SELECT STATEMENT) en zet ID 1 bovenaan de boom.

d) Een blad uit een boom kan altijd uitgevoerd worden. Een knoop die uit meerdere takken bestaat, kan alleen uitgevoerd worden als beide takken zijn uitgevoerd. Er wordt van links naar rechts gewerkt in de boom. Zet met deze regels in het achterhoofd de stappen in volgorde van uitvoering.

e) Schrijf code (pseudocode of PL/SQL) voor hoe deze zoekvraag uitgevoerd zal worden.

Extra) Zoals gezegd kan het plan dat je met een explain plan verkrijgt afwijken van het plan dat je in de "Row Source Operation" van het tkprof-bestand zie. Vraag het plan maar eens op van het UPDATE-commando uit hoofdstuk 3, die je uit het tkprof-bestand kunt halen. Waar komt het verschil vandaan?

5. Statistieken

Om de cost-based optimizer goede plannen te kunnen laten genereren, heeft hij actuele statistieken nodig van alle betrokken objecten. Tot en met versie 8 werd het ANALYZE commando gebruikt, maar vanaf versie 9 raadt Oracle het gebruik van package DBMS_STATS aan.

Lees Hoofdstuk 13 t/m paragraaf 13.3.1.2 (bladzijden 13-1 tot 13-7) van de Oracle Database Performance Tuning Guide over statistieken.

Maak de tabel opoo5 aan door script ***opoo5_installeer*** te draaien.

a) Draai een explain plan van "select nr from opoo5". Kloppen de geschatte cardinaliteiten? Hoe komt dat?

b) Draai script ***opoo5b***, zonder naar de inhoud van dit script te kijken. Draai weer een explain plan. Hoe verklaar je de getallen?

c) Bekijk nu de parameters van de procedure dbms_stats.gather_table_stats procedure in de Oracle Database PL/SQL Packages and Types Reference. Vergaar zelf statistieken op de tabel opoo5 en toon dat de getallen nu wel kloppen.

Extra) Bekijk de overige procedures van de dbms_stats package en krijg een idee wat er allemaal mogelijk is.

Verwijder tabel opoo5 door script ***opoo5_einde*** te draaien.

6. Optimaliseren

a) Bij de tabel KLANTEN worden attributen waarvan het verloop van hun waarde gedurende de tijd bijgehouden moet worden, opgeslagen als KLANT_PROFIELEN. Deze tabel kent een 20-tal van dit soort flexibele attributen. Een zoekvraag die alle klanten ophaalt inclusief de huidige waarde van 5 van de 20 flexibele attributen, moet geoptimaliseerd worden.

Draai het script ***opoo6a_installeer*** om de situatie aan te maken. In het script ***opoo6a*** staat de zoekvraag die geoptimaliseerd moet worden. Als je klaar bent, draai je ***opoo6a_einde*** om de installatie ongedaan te maken.

Tip: kijk of je het aantal tabeltoegangen kunt verminderen.

b) Er is een probleem met een zoekvraag in een applicatie. Nu duurt deze boven de seconde, wat onacceptabel is in een scherm. Aan jou het verzoek om de zoekvraag dusdanig te optimaliseren om deze vele factoren te versnellen.

Draai het script ***opoo6b_installeer*** om de situatie aan te maken. In het script ***opoo6b*** staat de zoekvraag die geoptimaliseerd moet worden. Als je klaar bent, draai je ***opoo6b_einde*** om de installatie ongedaan te maken.

Tip: gebruik je gezonde verstand om te zien of alles in de juiste volgorde wordt gedaan.

7. Histogrammen

Histogrammen dienen om fijnere statistieken te krijgen over de mogelijke waarden van een kolom. Zonder histogrammen gebruikt de cost based optimizer bij een zoekvraag als "select * from tabel where kolom = waarde" informatie als:

- NUM_ROWS – het aantal rijen in de tabel
- LOW_VALUE – de laagste waarde van de kolom
- HIGH_VALUE – de hoogste waarde van de kolom
- NUM_NULLS – het aantal rijen met een NULL-waarde voor de kolom
- NUM_DISTINCT – het aantal verschillende waarden (exclusief NULL)

Lees hoofdstuk 13.6.2 van de Oracle Database Performance Tuning Guide over histogrammen (bladzijden 13-23 tot 13-25). Draai script ***opoo7_installeer***.

a) Draai de zoekvraag uit het script ***opoo7*** en vraag het plan op met een explain plan commando. Kan je de formule achterhalen hoe deze cardinaliteit is berekend?

b) Het zou fijn zijn als deze zoekvraag gebruik maakt van de index op de Status-kolom. Maak daarom een frequentie-histogram aan om dit voor elkaar te krijgen. Vraag de kolom Histogram van de view USER_TAB_COL_STATISTICS uit om dit te controleren.

c) Maak een zoekvraag waarin je van een frequentie-histogram het volgende toont:
    - Een volgnummer van het bakje (1 t/m aantal bakjes (=bucket))
    - De waarde van het bakje
    - Het aantal rijen met deze waarde, oftewel de grootte van het bakje

    Tip: Gebruik de analytische functie LAG.

d) Maak van de eerder gemaakte histogram een hoogte-gebalanceerde-histogram. Bekijk het resultaat via de USER_HISTOGRAMS view. Bekijk de cardinaliteiten van de zoekvraag uit script opoo7, terwijl je de waarde veranderd van 0 naar 4.

8. Bindingsvariabelen en peeking

Om niet ieder SQL-commando iedere keer te hoeven parsen, worden alle cursoren die eenmaal geparsed zijn, bewaard in de library cache, onderdeel van de shared pool. Als je opnieuw een SQL-commando uitvoert en die al blijkt te bestaan, dan wordt het opgeslagen plan gebruikt. Bindingsvariabelen spelen een belangrijke rol in het optimaal hergebruiken van alle cursoren in de library cache. Zonder bindingsvariabelen zijn zoekvragen als "select * from emp where empno = 7839" en "select * from emp where empno = 7566" niet identiek en staan er twee cursoren in de library cache. Als bindingsvariabelen waren gebruikt dan was het SQL-commando "select * from emp where empno = :B1" geweest en stond deze maar eenmaal in de library cache. Wat betekent dat dit SQL-commando ook maar eenmaal geparsed hoeft te worden.

a) Bekijk de vier stukken tekst uit het script ***opoo8*** en onderzoek deze door ze door SQL*Trace/tkprof te halen. Welke variant gebruikt geen bindingsvariabelen?

b) Draai het script ***opoo8b_installeer***. Schrijf een script dat tabel OPOO8B tweemaal vult met 10.000 rijen: (1,'Omschrijving 1') t/m (10000,'Omschrijving 10000'). Eenmaal rij-voor-rij verwerking met een impliciete cursor (een INSERT … VALUES commando) en eenmaal EXECUTE IMMEDIATE zonder gebruik te maken van bindingsvariabelen. Onderzoek beide methodes en bepaal welke factor verschil er zit tussen de twee alternatieven. Als je klaar bent, verwijder je de tabel door ***opoo8b_einde*** te draaien.

Lees hoofdstuk 11.4.1.2 uit de Oracle Database Performance Tuning Guide over peeking (bladzijden 11-9 tot 11-10, dus tot "Estimating").

Tevens is het goed om te weten dat in versie 9 en 10 er nog geen sprake was van adaptieve cursor sharing. In deze versies werd altijd het plan gekozen van de allereerste keer dat de cursor werd geparst.

c) Draai het script ***opoo8c*** voor een demo van peeking en adaptieve cursor sharing. Druk steeds op een toets als je verder wilt.

9. PL/SQL bulkverwerking

PL/SQL bulkverwerking is HET alternatief voor grotere verwerkingsklussen die niet met SQL alleen opgelost kunnen worden. Je kunt hiermee het aantal executies van een DML-commando drastisch omlaag halen, wat weer leidt tot snellere verwerking. Begin met het draaien van script ***opoo9_installeer***.

a) Bekijk de code van het script ***opoo9a*** en schrijf op hoe vaak je verwacht dat het SELECT- respectievelijk het INSERT-commando een parse, execute en fetch opdracht zullen doen.

b) Zet tracing aan, draai het script ***opoo9a*** en maak en bekijk het tkprof-bestand. Waren dit de getallen die je verwachtte?

c) Lees in de Oracle Database Reference over de parameter PLSQL_OPTIMIZE_LEVEL. Zet deze met een ALTER SESSION op 0 en herhaal de stappen van vraag b.

d) Schrijf code zodanig dat:
   - het hetzelfde doet als script opoo9a: kopieren van t1 naar tabel t2.
   - het selecteren met een zelf in te stellen bulkgrootte werkt
   - de INSERT's in dezelfde porties worden uitgevoerd

   Gebruik hiervoor de LIMIT-clausule en het FORALL-statement. Voorbeelden hiervan zijn te vinden in hoofdstuk 12 van de Oracle Database PL/SQL Language Reference. Maak hiervan weer een tkprof-bestand om het verschil te zien.

Extra) Hoe had de code nog optimaler geschreven kunnen worden? Staaf dit met een tkprof-bestand.

Draai het script ***opoo9_einde*** om de gebruikte tabellen op te ruimen.

10. Gepartitioneerde tabellen

Partitionering wordt vaak als een stuk gereedschap gezien om performance mee op te krikken. In een OLTP-omgeving, is dit echter vrij lastig en zul je juist moeite moeten doen om dezelfde snelheid te behouden. Gegevenspakhuizen kunnen wel mooi profiteren van partitionering als de partities ze in staat stellen om slechts een klein deel van een grote tabel in een enkele partitie te selecteren. De volgende vragen zullen een aantal voor- en nadelen laten zien. Draai allereerst script ***opoo10_installeer***, om twee tabellen met dezelfde inhoud te maken: een gepartitioneerde tabel en een "normale" tabel.

a) Bekijk de scripts ***opoo10a1*** en ***opoo10a2*** en trace deze. Kan je het verschil verklaren? Wat kan je doen om de performance gelijk te houden? En wat is daar weer het nadeel van?

b) Verwijder alle rijen met een datum voor 1-1-2000, door de scripts ***opoo10b1*** en ***opoo10b2*** te draaien.

c) Schrijf een zoekvraag die de som van alle notabedragen in 2003 ophaalt van beide tabellen. Vraag de plannen op van beide zoekvragen en draai ze terwijl je met "set timing on" de doorlooptijd opvraagt.

Draai script ***opoo10_einde*** om de tabellen op te ruimen.