DOAG Konferenz 2016

Die Schlechten ins Kröpfchen - SQL Analyse für DBAs

Martin Klier
Managing Partner / Database Technology
Performing Databases GmbH

1. Inhalt und Zielgruppe

Der Vortrag zielt auf alle, die bereits erste Erfahrungen mit der Oracle Database sammeln konnten, und sich für die Analyse und Optimierung von Performancevorfällen interessieren.

Denn das Finden, Validieren und ggf. Verbessern von langsamem oder ressourcenintensivem SQL ist Teil der Aufgaben jedes DBAs, der seine/ihre Aufgaben ernst nimmt. Es kann schon spannend genug sein, entsprechende Statements zu identifizieren. Doch Verbesserungen für etwas vorzuschlagen oder auszuführen, dessen Zweck oft nicht ganz klar ist, ist eine ganz andere “Hausnummer”. Auf diese Art werden wir niemals Verbesserungen um Faktoren von 1 Million oder 100.000 erreichen - das ist ein Privileg der Designer und Anwendungsentwickler. Aber es gibt einen generischen Ansatz für uns "Datenbankflüsterer", um Statements anzupacken, die Aufruhr verursachen. Dann erreichen wir zwar nur Faktor 2, 5 oder 10 - mehr ist oft nicht drin, aber auch das ist immer einen Versuch wert!

Wir vermeiden in diesem Paper teure Tools, Voodoo-Features, aufeinander geschichtete Heuristiken und vor allem Vorgehensweisen, bei welchen ein Automatismus die Fehler des Vorangegangenen kompensiert. Ein unspektakulärer, erprobter Ansatz nach Logik und gesundem Menschenverstand.

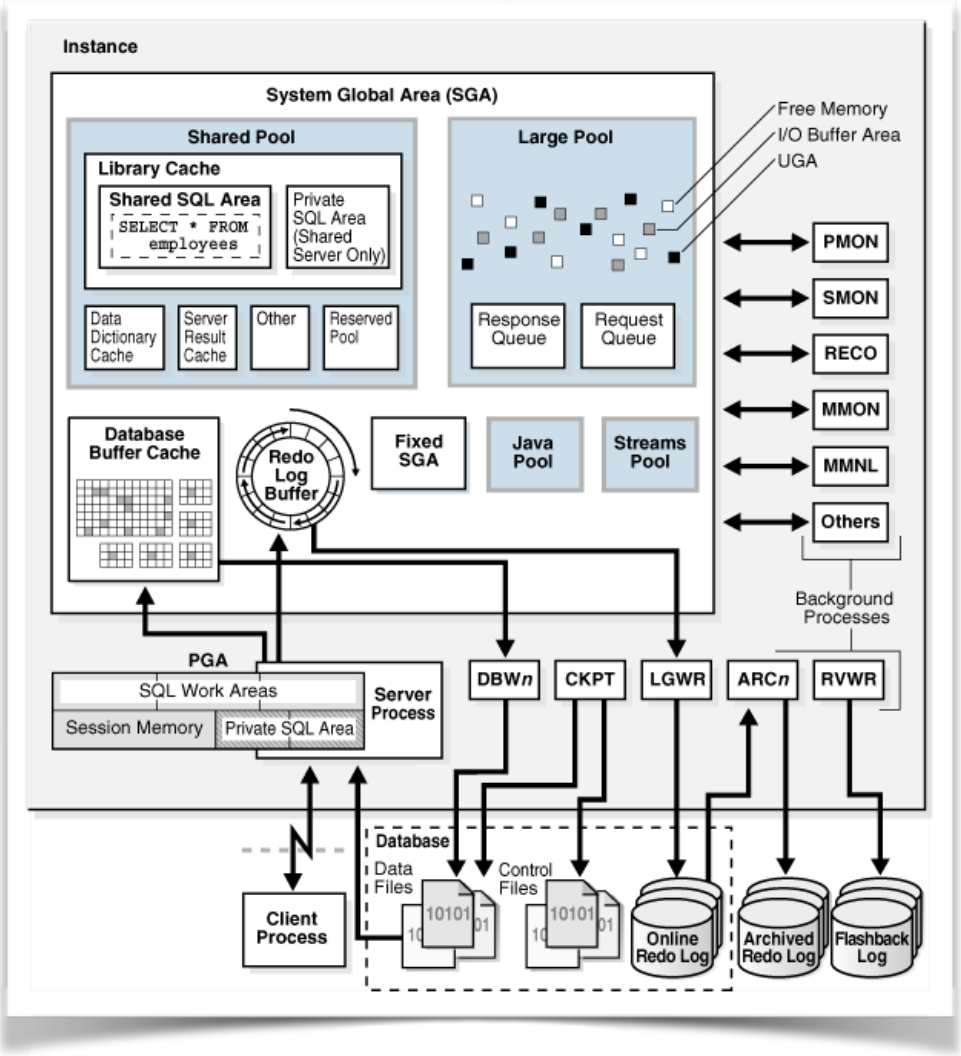
Bitte verwenden Sie als Unterlage zur Lektüre dieses Whitepapers meine Präsentation “The Bad One Into Your Crop - SQL Tuning Analysis for DBAs”, viele Fallbeispiele sind dort graphisch aufbereitet.

Durch begrenzten Raum werden einige Vorgänge deutlich vereinfacht dargestellt. Detailfragen, Kritik und Verbesserungsvorschläge sind jederzeit willkommen: Im persönlichen Gespräch, oder unter martin.klier@performing-db.com.

2. Basiswissen

Aufbau des RDBMS

Das Relationale Datenbank-Management-System (RDBMS) von Oracle besteht im Wesentlichen aus zwei Teilen: Während alle Komponenten im RAM, wie Speicherbereiche und Prozesse, zur “Instanz” zählen, bilden die Datafiles, Online Redo Logs und Control Files die eigentliche “Datenbank”. Diese Unterscheidung ist sinnvoll: Denn die Datenbank stellt das Ergebnis aller Bemühungen des RDBMS dar. Die Instanz ist “nur” das Werkzeug, das zum Ziel führt.

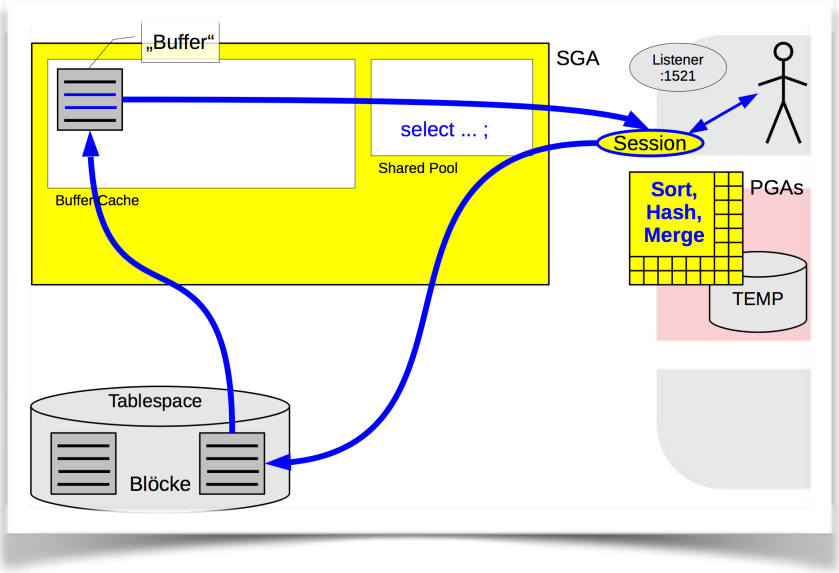


Diese Graphik aus der Oracle-Dokumentation 12cR1 zeigt das Zusammenwirken von Instanz und Datenbank.

Arbeiten mit dem Massenspeicher

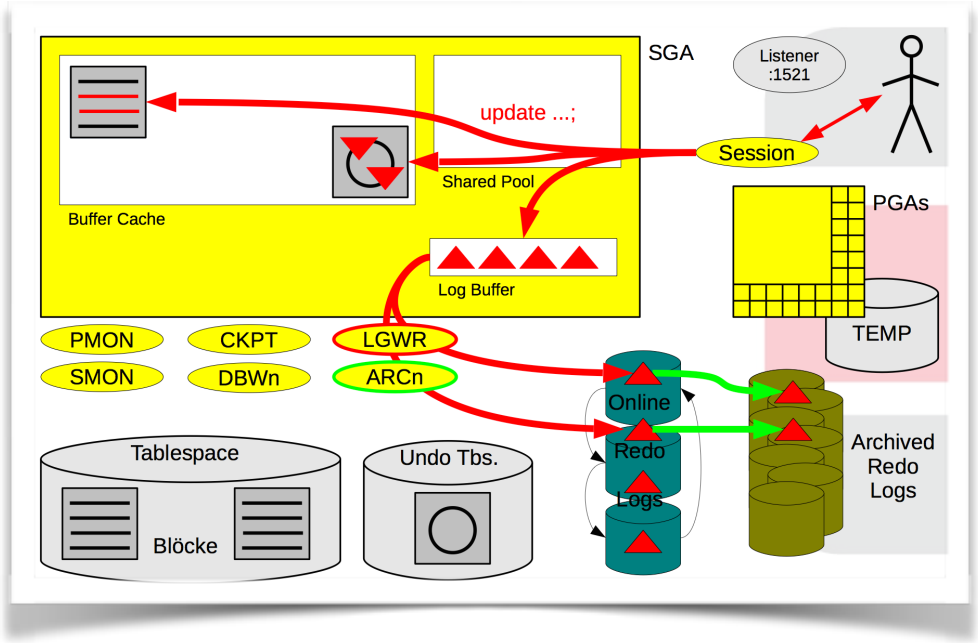
Zur Wiederholung sei kurz und vereinfacht die Handhabung von Daten im Oracle RDBMS dargestellt.

Lesen



Reguläre Lesevorgänge erfolgen durch den jeweiligen Serverprozess (“Session”) und bringen relevante Blöcke im Ganzen in den Buffer Cache. Ab hier spricht man nicht mehr vom Block, sondern vom Buffer. Von dort aus grenzt der Server auf Teilmengen/Rows ein.

Schreiben



- Verändernde Transaktionen schreiben zunächst auf den Buffer im Cache, erzeugen dort ebenso Undo-Informationen in einem weiteren Buffer.
- Für beide Vorgänge werden Redo-Daten im Redo Log Buffer eingetragen.

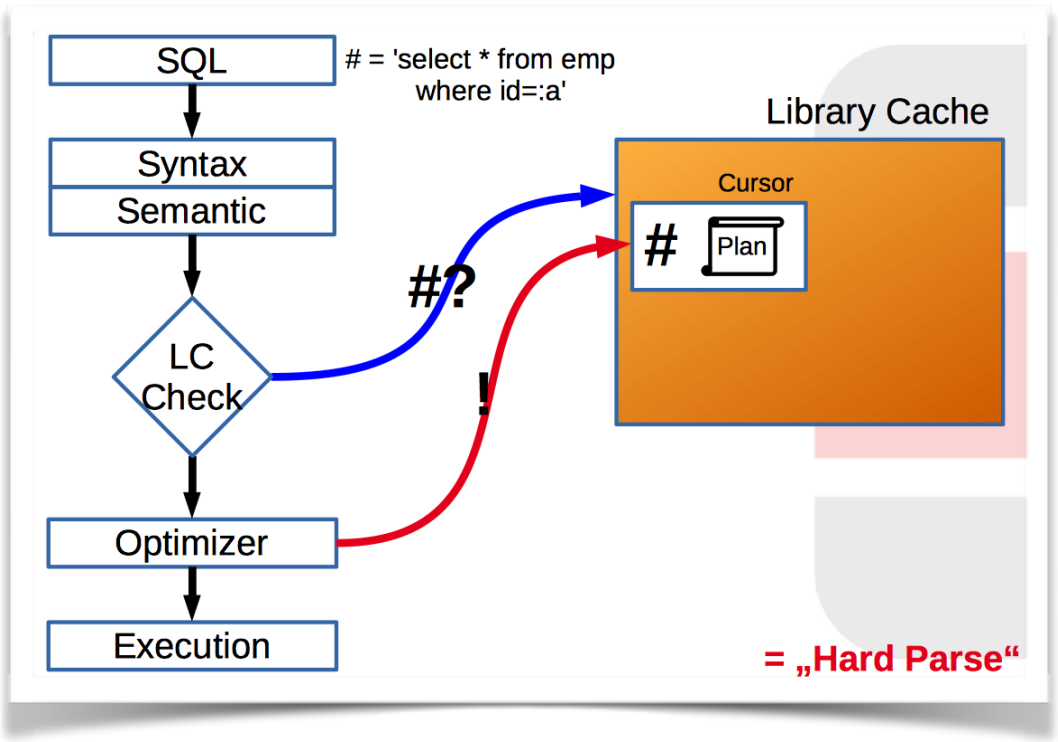
- Alle Buffer des Cache werden nach und nach durch den Komplex aus den Checkpoint- (CKPT) und Database Writer (DBWn) Prozesse in die Tablespace Datafiles synchronisiert.
- Der Log Writer Prozess (LGWR) schreibt die Inhalte des Redo Log Buffer in die Online Redo Logs. Sind alle Redo Logs voll und alle referenzierten Buffer synchronisiert, so werden die ältesten Einträge im Redo Log wieder überschrieben.
- Die aktuelle System Change Number wird im Control File aktualisiert, ebenso ggf. neue Datafile-Größen.
- Optional, aber sehr üblich für “Archivelog-Mode”: Die Archiver-Prozesse (ARCn) sorgen für die Duplizierung aller vollständig gefüllten Online Redo Logs in die sogenannten Archived Redo Logs (kurz: Archivelogs), die für verschiedene Recovery Szenarien benötigt werden, z.B. Vollständiges Recovery.
- Optional für “Flashback Database”, nur in Enterprise Edition: Der Recovery Writer Prozess (RVWR) kopiert angefallene Undo-Informationen in die Flashback Logs. Diese finden Verwendung um z.B. die ganze Datenbank in der Zeit zurückrollen zu können, ohne ein Backup einzuspielen.

3. Parsing und Optimierung

Über die Sessions eingehende Anweisungen liegen in hochsprachlicher Form als SQL vor. Diese Abstraktion ist vom DBMS in konkret ausführbare Anweisungen zu übersetzen. Daher wird jedes Statement einem Parsing- und gegebenenfalls auch einem Optimierungsprozess unterzogen. Nach der relativ unkomplizierten Prüfung der syntaktischen Korrektheit, validiert der Parser die Semantik: Stimmen Objektnamen, Felder, Datentypen und deren Zusammenhänge überein? Hierzu sind Zugriffe auf das Data Dictionary notwendig - das jedoch in aller Regel gut priorisiert in seinem Cache liegt.

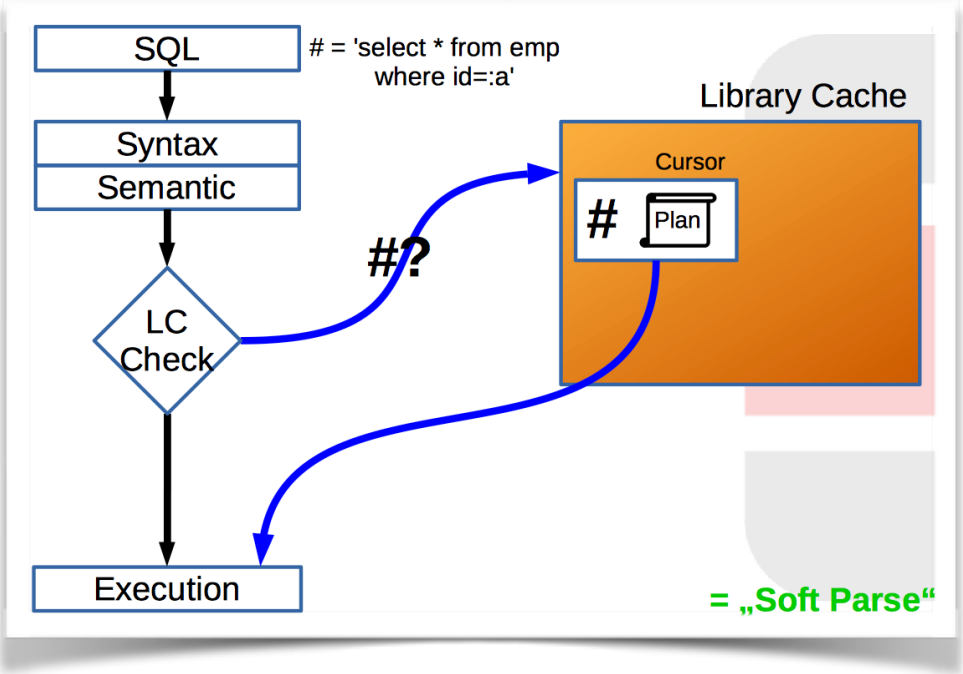
Cursor Sharing

Oracle hat schon vor langer Zeit erkannt, daß der eigentliche SQL Optimierungsvorgang (d.h. das maschinelle Finden des besten Weges, alle erforderlichen Daten in Zugriff zu nehmen) viele



Ressourcen verbraucht und daher möglichst selten stattfinden sollte. Dieses Feature nennt sich “Cursor Sharing”, und erlaubt unter Nutzung des Library Cache, bereits erstellte Ausführungspläne wiederzuverwenden. Es entsteht ein sogenannter Cursor, der durch den Hashwert des SQL Statements identifiziert und mit diesem als Schlüssel abgelegt wird. Er beinhaltet unter anderem den erstellten Ausführungsplan.

Beim sogenannten “Hard Parse” erfolgt der volle Durchgang des Optimizers, beim “Soft Parse” wird nach erfolgreichem “Library Cache Check” der vom ersten Lauf hinterlegte Plan zur Execution herangezogen.

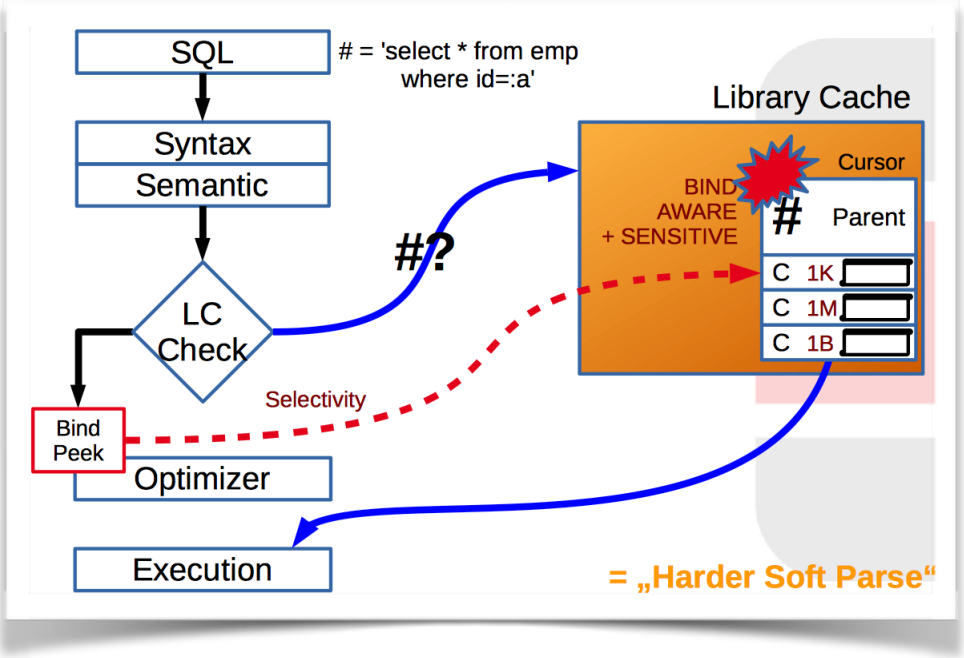


Voll zum Tragen kommen die Vorteile des Cursor Sharing erst, wenn variable Werte (die zu abweichenden Hashwerten führen und Cursor Sharing unmöglich machen würden) als sogenannte Bind-Variablen übergeben werden. Diese werden beim ersten Execute ausgewertet (Bind Peeking”) und fließen in die Optimierung wesentlich mit ein.

Dieses Verfahren hat jedoch einen gravierenden Nachteil: Wir sparen zwar einen Optimierungslauf ein, dieser nimmt jedoch an daß alle Werte dieser einen Variable die gleiche Selektivität aufweisen werden. Eine Nachprüfung findet nicht statt. Zu Illustration verweise ich auf meinen Blogartikel “Oracle Depends on Humidity” von 2012: (http://www.usn-it.de/index.php/2012/08/29/r-i-p-oracle-database-10g-oracle-depends-on-humidity/)

Adaptive Cursor Sharing

Ab Oracle 11.1 wurde für die nachträgliche Ermittlung der Selektivität einer Bindvariable ein Mechanismus eingebaut, der “Adaptive Cursor Sharing (ACS)”, also etwa “Sich anpassendes Cursor Sharing” genannt wurde. Nach einigen Anlaufschwierigkeiten stand er ab 11.2 zuverlässig zur Verfügung. Dieses ACS basiert auf “Statistics Feedback” (vor 12.1 hieß es noch Cardinality Feedback), nämlich der Rückmeldung, ob die aus den Statistiken gewonnenen Kardinalitäten auch den Ergebnissen des letzten Laufes entsprachen.



Falls nicht, wird der Plan als “Bind Aware” gekennzeichnet und bei der nächsten Ausführung dieses SQL Statements die Bindvariable erneut ausgewertet und ein neuer Ausführungsplan für die ermittelte Selektivitätsklasse der Variablen hinterlegt. Nun ist der Plan auch “Bind Sensitive”, und bei jeder neuen Ausführung wird immer mindestens die Selektivität der Bindvariable hinterfragt (“Bind Peeking”). Das ist nun etwas teurer als der klassische “Soft Parse”, und trägt daher den Untergrundnamen “Harder Soft Parse”.

4. Mein Weg der Analyse

Ich möchte Ihnen nach dieser Einführung in die Grundlagen gerne meinen Weg zeigen, an die Analyse heranzugehen. Das muß nicht jedermanns Geschmack sein, und ist sicher auch nicht perfekt. Aber es hat seine Berechtigung, und passt zu meiner generellen Denk- und Arbeitsweise.

Persönlich

Das ist ein wichtiger Punkt: Ein Performance-Experte darf und soll seinen eigenen Stil entwickeln, denn den universell “richtigen Weg” gibt es nicht. Spielen Sie keine angenommene Rolle, sondern sind Sie Sie selbst, stehen Sie zu ihren Methoden und Ergebnissen. Jedoch würde ich Lernfreude, Ergebnisoffenheit, Faktenwissen und die Fähigkeit, Ergebnisse verständlich darzustellen als allgemein äußerst nützlich einstufen.

Analyse beginnt stets mit der Frage: “Was ist das Problem?” Denn nur wenn der Beschwerdeführer abgeholt und verstanden wird, kann sich überhaupt eine Ursache oder Lösung finden, die dieser auch anerkennt. Womit man dann beginnt, ist eine Frage der Methodik, Erfahrung und nicht zuletzt von Instinkt. Letzteren hat jeder Rennfahrer, Reiter, Segler und Flieger: Das gute alte “Po-Gefühl” für sein Arbeitsgerät. Wenn ich nun zum ersten Mal eine bis dato unbekannte Umgebung “sattle”, ist es das erste Bestreben, diese Verbindung aufzubauen, und mit allen Sinnen einzutauchen in die Umweltbedingungen, die dort herrschen. Das klingt vielleicht etwas pathetisch, trifft es aber ziemlich genau.

Instinkt ist jedoch “nur” der persönliche Weg. Er fördert im Durchschnitt zwar gepaart mit etwas Erfahrung erheblich die Effizienz, und beschleunigt damit den Weg zu einer Lösung. Instinkt hat aber in Begründungen, Ergebnissen und Schlussfolgerungen nichts verloren. Entscheidungen haben auf soliden Fakten und Hintergrundwissen zu basieren.

Tools

Meine Liste an empfohlenen Tools für die Analyse von Oracle Datenbank Performance ist erfreulich kurz:

- AWR + ASH oder Statspack
- SQL*Plus + rlwrap oder SQLcl
- SQL Developer (man muß nicht puristischer sein als nötig)
- dbms_xplan.display_cursor()

Dazu kommt dann fallabhängig für schnellen Überblick und effiziente Kommunikation von Zahlen noch die Datenaufbereitung und Visualisierung z.B. mit Splunk, R oder im Notfall eine Tabellenkalkulation mit Charting-Funktion.

5. Der Störenfried

Das oder die Ursachen zu finden, die Wesentlich zur beklagten Situation beitragen, ist selten einfach, und hängt stark vom umgebenden Business Case ab. Unter der Annahme, daß tatsächlich ein oder mehrere SQL Statements verantwortlich sind, bietet sich die Unterscheidung von zwei Fällen an:

a) Ein Statement sättigt eine Ressource so, daß die Gesamtleistung beeinträchtigt wird. In dem Fall führen alle Methoden zur Lastanalyse zum Ziel (z.B. AWR). Hier ist der DBA zu Hause.
b) Ein Einzelvorgang dauert zu lange, ist aber in der Gesamtbelastung zu vernachlässigen. Hier greift nur der Ansatz über die Applikation, über ein auf Debugging oder Code Instrumentierung basierendes Profiling. Erst dann setzt die datenbankbasierte Fehlersuche effektiv ein, kann aber vorher schon über das Ausschlussverfahren nützliche Hinweise zum Gesamtbild beitragen (z.B. Auflistung relevanter SQLs mit Laufzeit > x).

In beiden Fällen sind die hier vorgestellten Ansätze in unterschiedlicher Ausprägung nützlich.

AWR Reports auswerten

Ich setze im Folgenden die Kenntnisse zum Erzeugen eines AWR Reports für einen bestimmten Zeitraum voraus. Liegt dieser dann vor, so ist zunächst die Relation zwischen Systemleistung und Belastung im betrachteten Zeitraum interessant. Es handelt sich hier um eine von vielen möglichen Szenarien, es ist nur dazu gedacht, den Ansatz zu vermitteln.

Überblick

Wie viel Zeit betrachten wir (Elapsed Time)? z.B. 1h = 60min = 3600s
Wie viele CPU Cores standen zur Verfügung? z.B. 12 = 720 CPU-Minuten
Wie viel Zeit hat die DB als Arbeitszeit aufgezeichnet (DB Time)? z.B. 52min

Hier scheidet auf den ersten Blick globale CPU-Überlast aus (52min von 720min), allerdings entspricht 60min den 100% eines Cores, und hier liegen wir schon dichter daran.

“Report Summary” liefert in Form von User Calls, SQL Executes und Transactions (jeweils Durchschnitte pro Sekunde) einen wichtigen Eindruck von der Last in diesem Zeitraum.

Einordnung

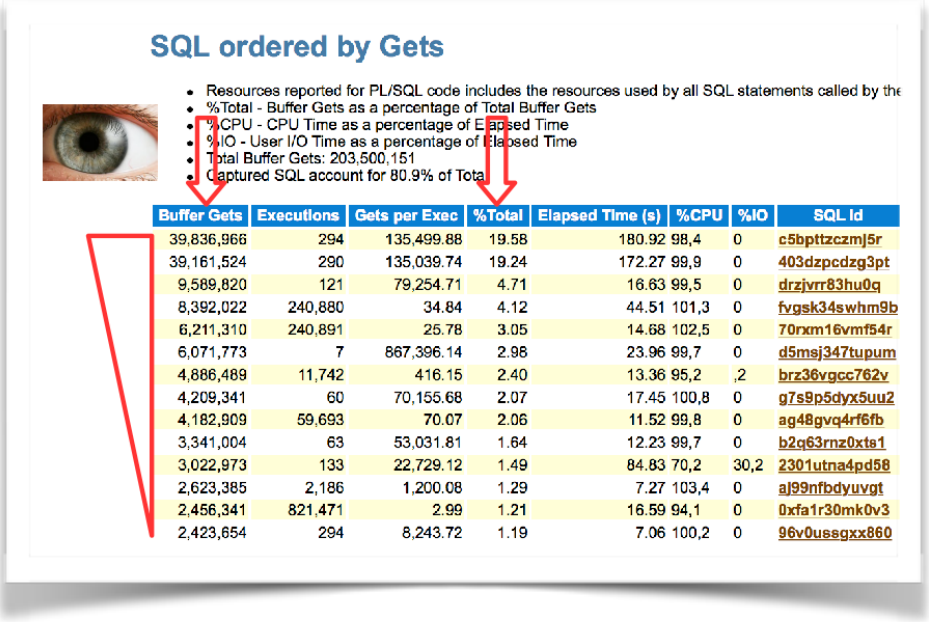
Im “Report Summary” lohnt sich im weiteren Verlauf eine gewisse Kategorisierung nach IO- und CPU-Anteilen. So ordne ich Logical Reads, Block Changes, Parses und Hard Parses uneingeschränkt der Kategorie “CPU” zu: Speicherzugriffe sind nun mal CPU-Arbeit.
Physical Read/Write, Read/Write IO requests, Read/Write IO gehören ebenso klar zum IO Teil.

Die “Top 10 Foreground Events” liefern weitere Relationen: Zu wie viel Prozent (von oben: “DB Time”) haben wir CPU genutzt? z.B. 95% von 52 = 49,5min
Wieviel % IO? z.B. 9%
Wie viele Serialisierungen (Latches, Mutextes etc.)? z.B. keine

Bitte beachten Sie, daß manche Waits CPU Zeit beinhalten, so z.B. “log file sync” oder Mutex waits. Dadurch addieren sich die meisten Fälle auf mehr als 100% auf. Ich möchte dazu auf die vorzüglichen Detailanalysen zum Wait Interface und seinen Messfehlern von Frits Hoogland hinweisen.

Fallabhängige Recherche I

Durch die oben beispielhaft verwendeten Zahlen deutet vieles auf hohe Beanspruchung der CPU ohne Serialisierungsprobleme hin. Damit rücken die Buffer Gets in den Fokus, und SQLs mit vielen Buffer Gets zeigt AWR sehr zuverlässig an.



SQL ordered by Gets

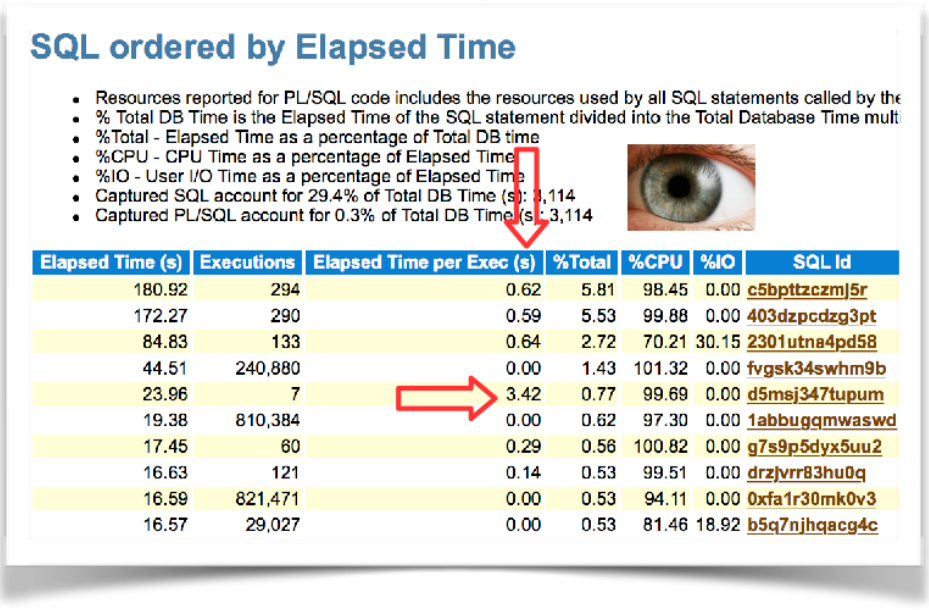
- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the
- %Total - Buffer Gets as a percentage of Total Buffer Gets
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Total Buffer Gets: 203,500,151
- Captured SQL account for 80.9% of Total

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id
39,836,966	294	135,499.88	19.58	180.92	98,4	0	c5bpttzczmj5r
39,161,524	290	135,039.74	19.24	172.27	99,9	0	403dzpcdzg3pt
9,589,820	121	79,254.71	4.71	16.63	99,5	0	drzjvrr83hu0q
8,392,022	240,880	34.84	4.12	44.51	101,3	0	fvgsk34swhm9b
6,211,310	240,891	25.78	3.05	14.68	102,5	0	70rxm16vmf54r
6,071,773	7	867,396.14	2.98	23.96	99,7	0	d5msj347tupum
4,886,489	11,742	416.15	2.40	13.36	95,2	,2	brz36vgcc762v
4,209,341	60	70,155.68	2.07	17.45	100,8	0	g7s9p5dyx5uu2
4,182,909	59,693	70.07	2.06	11.52	99,8	0	ag48gvq4rf6fb
3,341,004	63	53,031.81	1.64	12.23	99,7	0	b2q63rnz0xte1
3,022,973	133	22,729.12	1.49	84.83	70,2	30,2	2301utna4pd58
2,623,385	2,186	1,200.08	1.29	7.27	103,4	0	aj99nfbdyuvgt
2,456,341	821,471	2.99	1.21	16.59	94,1	0	0xfa1r30mk0v3
2,423,654	294	8,243.72	1.19	7.06	100,2	0	96v0ussgxx860

Zusammengeführt mit dem Problem (“Ist eines der genannten SQLs relevant?”) liefert dieser Ansatz schon einmal eine oder mehrere SQL_IDs die näher untersucht werden sollten.

Fallabhängige Recherche II

Außerdem sind natürlich auch die Ausführungszeiten interessant - einmal ganz offensichtlich natürlich für den Endanwender, falls die Applikation in hohem Maße von der Antwortzeit eines Einzelstatements abhängt. In jedem Fall sind jedoch das genau die Zeiten, die auch im Profil der Applikation wiederzufinden sind.



SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time mult
- %Total - Elapsed Time as a percentage of Total DB time
- %CPU - CPU Time as a percentage of Elapsed Time
- %IO - User I/O Time as a percentage of Elapsed Time
- Captured SQL account for 29.4% of Total DB Time (s): 3,114
- Captured PL/SQL account for 0.3% of Total DB Time (s): 3,114

Elapsed Time (s)	Executions	Elapsed Time per Exec (s)	%Total	%CPU	%IO	SQL Id
180.92	294	0.62	5.81	98.45	0.00	c5bpttzczmj5r
172.27	290	0.59	5.53	99.88	0.00	403dzpcdzg3pt
84.83	133	0.64	2.72	70.21	30.15	2301utna4pd58
44.51	240,880	0.00	1.43	101.32	0.00	fvgsk34swhm9b
23.96	7	3.42	0.77	99.69	0.00	d5msj347tupum
19.38	810,384	0.00	0.62	97.30	0.00	1abbuggmwaswd
17.45	60	0.29	0.56	100.82	0.00	g7s9p5dyx5uu2
16.63	121	0.14	0.53	99.51	0.00	drzjvrr83hu0q
16.59	821,471	0.00	0.53	94.11	0.00	0xfa1r30mk0v3
16.57	29,027	0.00	0.53	81.46	18.92	b5q7njhqacg4c

Das konkrete Beispiel zeigt den Abschnitt “SQL ordered by Elapsed Time”. Hier ist ein Statement zu finden, das im Schnitt 3,42 Sekunden für eine Ausführung benötigt, und sieben mal ausgeführt wurde. Sollte das interaktiv laufen, und die Applikation einen gewissen Echtzeitanspruch haben, kann und wird es zu Beschwerden führen. Auch für dieses lohnt es sich, die SQL_ID für die Untersuchung vorzumerken.

Weitere Tips

Folgende Punkte empfehle ich noch zur weiteren Prüfung - als “Joker” oder um auch bei nicht ganz alltägliche Problemchen die Analyse erfolgreich abzuschließen:

- Anzahl SQL Executions (denke: Software “läuft so schnell sie kann”)
- Shared Memory Nutzung / Version Count (z.B. durch Child Cursor Probleme)
- Plausibilität Instanzparameter prüfen
- NLS Einstellungen ansehen und bei der weiteren Analyse berücksichtigen (Indexnutzung, Sortierung, Stringvergleiche etc.)
- Advanced Queues und (drei Arten von) Jobs prüfen

Ausführungspläne lesen

ist der “Störenfried” gefunden, geht es darum diesen im Detail zu untersuchen. Ein wichtiges Werkzeug dafür ist der Ausführungsplan (“Execution Plan”). Alle Ausführungspläne im Folgenden wurden nach meiner Lieblingsmethode erzeugt:

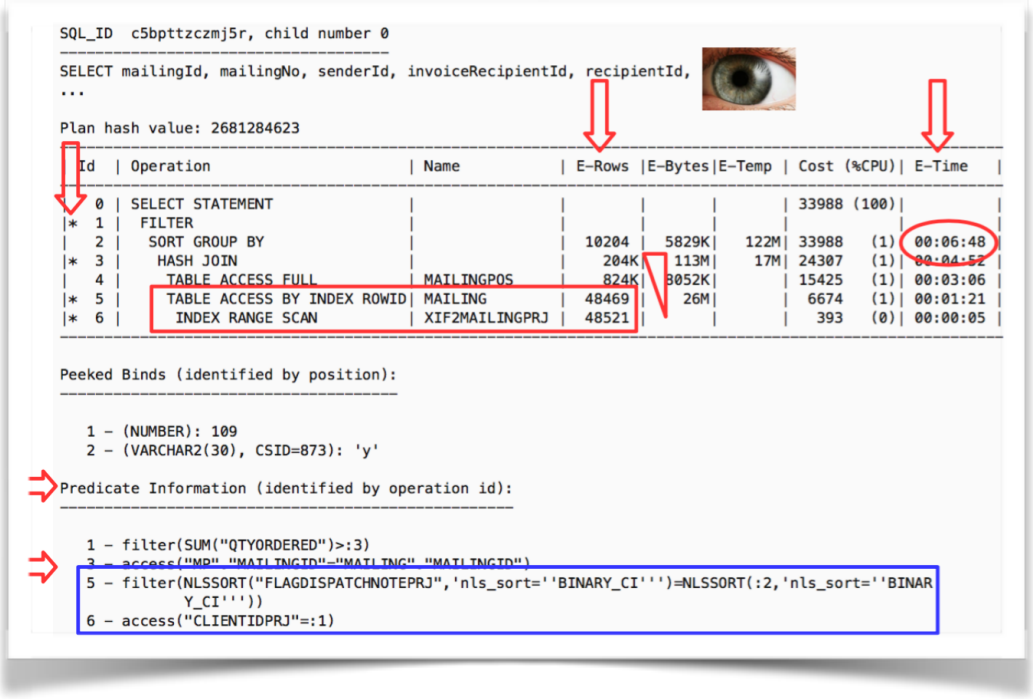
```
SELECT *
FROM table(
                        DBMS_XPLAN.DISPLAY_CURSOR(
                                '&&SQL_ID',
                                null,
                                'COST,IOSTATS,LAST,ADVANCED,ADAPTIVE'
                                )
                        );
```

Fallabhängige Recherche

Das erste Beispiel, das ich bearbeiten möchte, zeigt einen Ausführungsplan mit hohen Kosten (rund 34.000), der im ersten Beispiel zu “Lesen von AWR Reports” durch sehr viele Buffer Gets aufgefallen ist. Die “Plausibilitätsprüfung” hat ebenfalls ergeben, daß das Statement relativ langsam läuft, etwa 5 Sekunden wurden uns genannt.

Läuft das DB System auf STATISTICS_LEVEL=TYPICAL, müssen wir im Execution Plan leider auf die Auswertung tatsächlicher Buffer Gets und realer Kardinalität verzichten. Ich würde dem stets den Vorzug geben, aber durch seine Auswirkungen auf Laufzeit und AWR Größe ist STATISTICS_LEVEL nicht in allen produktiven Umgebungen veränderlich - außer in Zeiten allerhöchster Not. Aber auch anhand der Estimated Rows und Estimated Bytes ist mein Ansatz anschaulich zu demonstrieren. Auf der Suche nach dem bedeutendsten Lastverursacher, findet sich zunächst einmal der hash Join in Zeile 3. Dieser wiederum nährt sich vorrangig aus dem Table Access by Index Rowid in Zeile 5. (Und nicht, wie man zunächst vielleicht beanstanden möchte, aus dem Table Access Full in Zeile 4!)

Die Zeilen 5 und 6 lösen beim Zugriff auf die Tabelle MAILING und einen ihrer Indexes zwei Prädikate auf. Dies erkennen Sie an den Asterisken (*) vor der Zeilennummer. Im Bereich “Predicate Information” weiter unten (blaue Markierung) finden Sie die Gleichungen, die an der Stelle evaluiert werden. Wir sehen im konkreten Fall die CLIENTPRJID gegen die erste Bind



```
SQL_ID  c5bpttzczmj5r, child number 0
-------------------------------------
SELECT mailingId, mailingNo, senderId, invoiceRecipientId, recipientId,
...

Plan hash value: 2681284623
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name           | E-Rows |E-Bytes|E-Temp | Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |        |       |       | 33988 (100)|          |
|*  1 |  FILTER                       |                |        |       |       |            |          |
|   2 |   SORT GROUP BY               |                |  10204 |  5829K|  122M| 33988   (1)| 00:06:48 |
|*  3 |    HASH JOIN                  |                |    204K|   113M|   17M| 24307   (1)| 00:04:52 |
|   4 |     TABLE ACCESS FULL         | MAILINGPOS     |    824K|  3052K|       | 15425   (1)| 00:03:06 |
|*  5 |     TABLE ACCESS BY INDEX ROWID| MAILING       |  48469 |   26M|       |  6674   (1)| 00:01:21 |
|*  6 |      INDEX RANGE SCAN         | XIF2MAILINGPRJ |  48521 |       |       |   393   (0)| 00:00:05 |
-----------------------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - (NUMBER): 109
   2 - (VARCHAR2(30), CSID=873): 'y'

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(SUM("QTYORDERED")>:3)
   3 - access("MP"."MAILINGID"="MAILING"."MAILINGID")
   5 - filter(NLSSORT("FLAGDISPATCHNOTEPRJ",'nls_sort=''BINARY_CI''')=NLSSORT(:2,'nls_sort=''BINAR
              Y_CI'''))
   6 - access("CLIENTIDPRJ"=:1)
```

Variable, und FLAGDISPATCHNODEPRJ im Rahmen einer NLSSORT() Funktion gegen die zweite Bind Variable. Diese Funktion bewirkt mit dem Parameter BINARY_CI, daß das angegebene Textfeld Case Insensitive (CI), d.h. ohne Berücksichtigung von Groß- und Kleinschreibung, durchsucht wird.

Ganz offensichtlich ist jedoch CLIENTPRJID alleine nicht sehr selektiv, so daß der Großteil der Rohdaten gegen die NLSSORT() Funktion geprüft werden muß. Das bedeutet viele Buffer gets aus dem Cache (oder db file sequential read von Disk) plus erhebliche CPU-Belastung.

Lösung

Um das beschriebene Problem zu lösen, empfiehlt sich hier zunächst die Prüfung der Selektivität verwendeter Bind Variablen, wie sie “Peeked Binds” angibt. Doch Vorsicht, hier handelt es sich nur um die Inhalte der beiden Variablen, die zur Erzeugung des Plans geführt haben. Das muß nicht repräsentativ sein, und es empfiehlt sich, das System dahingehend weiter zu untersuchen.

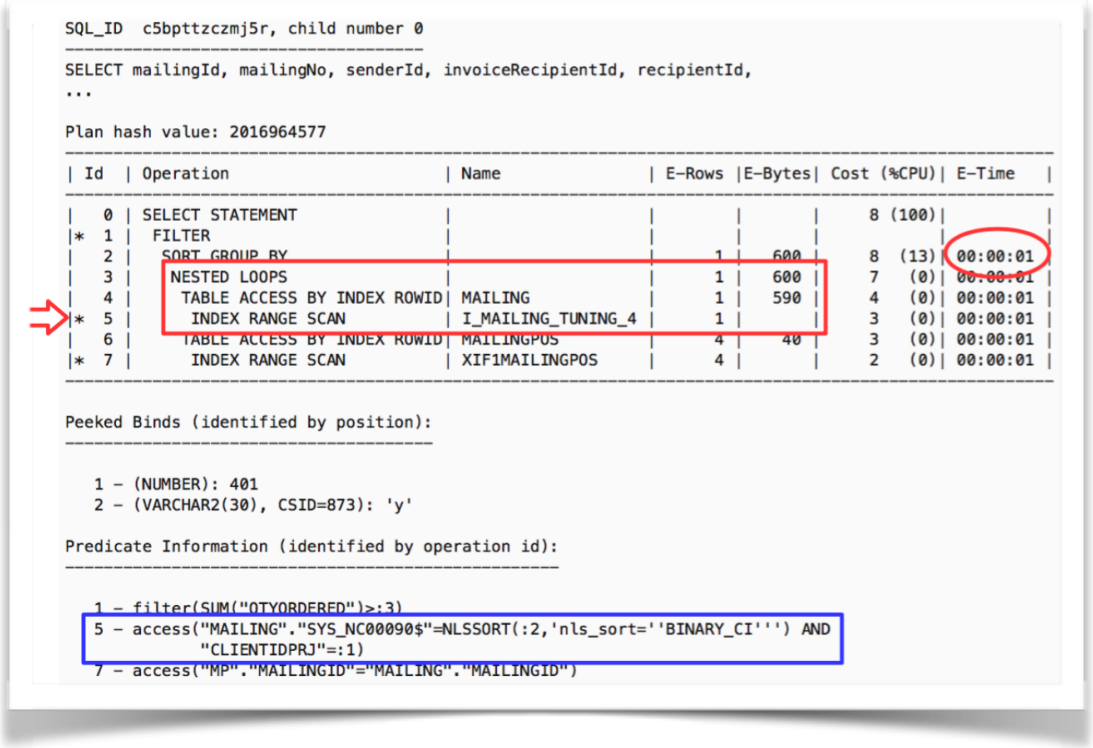
Ist man sich jedoch hinreichend sicher, daß FLAGDISPATCHNODEPRJ die bessere und CLIENTPRJID die zweitbeste Einschränkung auf Tabelle MAILING ist, so bietet sich folgender mehrspaltiger Function Based Index zur Behebung des Problems an:

```
Create index I_MAILING_TUNING_4 on MAILING (
       NLSSORT("FLAGDISPATCHNOTEPRJ",'nls_sort=''BINARY_CI'''),
       CLIENTIDPRJ
       );
```

Danach sollten unbedingt neue Statistiken auf die Tabelle und ihre abhängigen Objekte erzeugt werden - ich empfehle in der Regel für alle “Skewed” Columns auch Histogramme anzulegen.

Natürlich sollte ein solcher Eingriff in das Schema der Applikation von einem Chnageprozess begleitet werden. Mindestens umfasst dieser die Gegenprüfung, ob sich ein ähnliches Statement nun “besser” im Sinne der Performanceentfaltung verhält.

Diese Kalkulation geht hier auf: Beide Prädikate werden nun in Zeile 5 über einen INDEX RANGE SCAN evaluiert, und die Kosten fallen auf 8 (von 34.000). Der Optimizer scheint nun bezirzt - doch



```
SQL_ID  c5bpttzczmj5r, child number 0
-------------------------------------
SELECT mailingId, mailingNo, senderId, invoiceRecipientId, recipientId,
...

Plan hash value: 2016964577
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | E-Rows |E-Bytes| Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |        |       |     8 (100)|          |
|*  1 |  FILTER                       |                     |        |       |            |          |
|   2 |   SORT GROUP BY               |                     |      1 |   600 |     8  (13)| 00:00:01 |
|   3 |    NESTED LOOPS               |                     |      1 |   600 |     7   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| MAILING             |      1 |   590 |     4   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN         | I_MAILING_TUNING_4  |      1 |       |     3   (0)| 00:00:01 |
|   6 |     TABLE ACCESS BY INDEX ROWID| MAILINGPOS          |      4 |    40 |     3   (0)| 00:00:01 |
|*  7 |      INDEX RANGE SCAN         | XIF1MAILINGPOS      |      4 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - (NUMBER): 401
   2 - (VARCHAR2(30), CSID=873): 'y'

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(SUM("QTYORDERED")>:3)
   5 - access("MAILING"."SYS_NC00090$"=NLSSORT(:2,'nls_sort=''BINARY_CI''') AND
              "CLIENTIDPRJ"=:1)
   7 - access("MP"."MAILINGID"="MAILING"."MAILINGID")
```

wie sieht die Realität aus? Diese lässt sich nur über die SQL Ausführungsstatistiken ermitteln, und hier konnten die durchschnittlichen Buffer Gets pro Execution von 135k auf 1k reduziert werden. Das ist noch immer nicht brilliant, aber schon deutlich besser. Die Ausführungszeit fiel von 6:48s auf 0:01s, damit war der Fall für den Kunden erledigt.

Die Nachsorge erstreckte sich dann auf die Überprüfung der Planstabilität, wofür später noch Extended Statistics für die Kombination der beiden Spalten angelegt wurden. Damit erkennt der Optimizer deren gemeinsame Kardinalität.

6. Tips

Nun noch kurz vor Schluss einige Ratschläge aus der Praxis für die Praxis.

- Der nächste Crash / die Eskalation / die Krise kommt bestimmt. Sorgen Sie vor, indem Sie “Ihre” Systeme regelmäßig auf anfällige Statements überprüfen. Statements, die viele Ressourcen (z.B. Buffer Gets oder IOs) benötigen, reagieren empfindlich auf deren Verknappung. Die nächste Beschwerde ist dann absehbar. Sorgen Sie vor, schaffen sie vermeidbare Baustellen schon “zu Friedenszeiten” aus dem Wege.
- Einer weiß es immer besser: Die Kollegen, die Vorgesetzten, ein Berater. Setzen Sie sich in Ruhe (auch wieder “zu Friedenszeiten”) hin, und entwickeln Sie ihren eigenen Stil. Bereiten Sie Ihre Tools vor, werden Sie damit vertraut. Ob Sie selbstegebaut oder gekauft, frei oder kostenpflichtig sind, ist völlig unerheblich: Sie müssen für das passen was Sie können und erreichen wollen. Aber Sie sollten vertraut damit sein. Dann können Sie Ihre eigenen Ergebnisse erklären, und Ihre Herangehensweise begründen. Fehler liegen in der Natur der Sache, aber mit jedem Fehler werden Sie besser.
- Messen ist immer ein Vergleich. Meine klare Empfehlung ist, alles auf die Sekunde herunter zu brechen. Die Faktoren 3600 auf die Stunde und 86.400 für einen Tag sehen zunächst sperrig aus, aber sie gehen ins Blut über. Aber IOs, FLOPs, und hundert andere Referenzwerte der Hardware (und oft auch der Software) nutzen die Sekunde als Basis. Lassen Sie sich davon anstecken, es fördert die Transparenz und die Verständlichkeit.
- Wissen ist Macht, doch nur Beweise sichern die Weltherrschaft. Schön wäre es - aber Fakt ist auch, daß Wissen ohne Beweis allenfalls am Stammtisch oder beim Mittagessen Wert hat. Spätestens beim Budget hört meist der Spaß schon auf, und wenn in einer Krise schmerzhafte Maßnahmen angesagt sind, möchte jeder Manager mehr in der Hand haben als das erprobte Bauchgefühl seines Datenbankexperten. Lernen Sie daher, von der Aufnahme eines Falls bis zur Lösung alles zu dokumentieren. Damit sichern Sie wichtige Beweismittel über die erreichte Verbesserung und die getroffenen Entscheidungen. Ob Sie Eintragungen in ein Ticketsystem, Textfile, Wiki oder Notizsystem machen, ist unerheblich. Aber eine klar aufgezeichnete Maßnahmenkette lässt sich nicht mehr plausibel widerlegen, und ist ab dann mindestens als ein Zweig der Wahrheit zu berücksichtigen. Auch der vielgeschmähte Screenshot hat trotz im Überfluss verfügbarer Bildbearbeitungssoftware noch immer seinen Wert.

7. Abschluß

Das komplexe Thema wie auch dieses Dokument kennen kein Ende, nur die Aufforderung an alle Leser, sich weiterzubilden, und die “losen Enden” der Geschichte aufzugreifen, als Ansatzpunkt für das nächste Projekt, oder die eigene Fortbildung aus Interesse.

Für jeden, der mehr über das Oracle RDBMS erfahren möchte, sei dessen vorzügliche Onlinedokumentation empfohlen, für die 12c derzeit hier zu finden:
http://docs.oracle.com/database/121/index.htm

8. How to get in touch

Martin Klier
Managing Partner / Database Technology
at Performing Databases

Wiesauer Strasse 27
95666 Mitterteich
GERMANY

Phone: +49 9633 631
Fax: +49 9633 4199
Mobile: +49 170 PERFDB7 (+49 170 7373327)

mailto:martin.klier@performing-db.com

Twitter: @MartinKlierDBA

About us

We are "**performing databases**" - and we are keen on caring for you:
"**Your reliability. Our concern.**"

Specialized in services around database technology, supreme performance and the highest availability of your systems is our ultimate and most important goal.

Performing Databases GmbH
Your reliability. Our concern.

http://www.performing-databases.com