Oracle Optimizer System Statistiken Update 2018

Autor: Randolf Geist, oracle-performance.de

Seit einigen Jahren arbeitet der Oracle Optimizer, der dafür zuständig ist, SQL in einen ausführbaren Plan umzusetzen, mit sogenannten System Statistiken im Rahmen der Kostenabschätzung, die zur endgültigen Planauswahl herangezogen wird. Dieser Artikel geht auf die Grundlagen dieser System Statistiken ein, erklärt deren Historie und zeigt auf, was sich in den aktuellen Versionen 12c und 18c diesbezüglich getan hat. Weiterhin wird auch auf die mit Oracle 11.2 eingeführte I/O Kalibrierung eingegangen, die eine sehr ähnliche Funktionalität bietet.

System Statistiken sind verschiedene Messwerte, die Auskunft über die CPU- und I/O-Leistung eines Systems geben. Diese werden vom Oracle Optimizer dazu verwendet, die Kosten für Index- und Tabellenzugriffe zu berechnen.

Historie der System Statistiken

Seit der Einführung des kostenbasierten Optimizer (Cost Based Optimizer, CBO) in der Version 7 geschieht die Auswahl des auszuführenden Plans – welcher eine Umsetzung der 4GL-Anweisung in Form von SQL, das nur beschreibt, was man sehen / tun möchte, aber nicht wie, in eine prozedurale 3GL darstellt – über eben die berechneten Kosten, und bis auf wenige Ausnahmen wird der Plan mit den geringsten Kosten ausgewählt.

„Kosten“ bedeuten in diesem Zusammenhang maßgeblich die Anzahl der notwendigen I/Os – das Kostenmodell des CBOs beruht auf der Annahme, dass jeglicher Zugriff auf einen Block I/O notwendig macht - eventuelles Caching im Buffer Cache wird hier also wohl aus Konsistenzgründen nicht berücksichtigt - und die Einheit der berechneten Kosten ist der Einzelblockzugriff („Single Block Read“), also der Zugriff auf einen einzelnen Block.

Insofern stellt der CBO eine Art Compiler da, der SQL in einen ausführbaren Pseudocode übersetzt. Dabei entsteht eine Menge möglicher Code-Alternativen, welche über die Kostenabschätzung bewertet werden.

Wesentliche Anteile der berechneten Kosten beruhen auf Mengenabschätzungen und wie kostspielig die dazugehörigen Operationen sind, zum Beispiel wie häufig eine Schleife - als konkretes Beispiel sei ein Nested Loop Join genannt - durchlaufen werden muss, und wie aufwändig eine solche Iteration geschätzt wird, was dann als

Multiplikation aus Anzahl Iterationen und Kosten pro Iteration die Gesamtkosten einer solchen Operation ergibt. Was hierbei häufig zu Irritationen führt und daher wichtig zu verstehen ist, dass insbesondere diese Mengenabschätzungen, also beim genannten Beispiel, wie häufig die Schleife durchlaufen werden muss, aus verschiedensten Gründen sehr leicht falsch sein können, und daher der CBO Pläne auswählen kann, die sich in der Realität als nicht effizient erweisen. Grundsätzlich kann man hier beim heutigen Stand des CBOs davon ausgehen: Wären die Abschätzungen korrekt, wäre der ausgewählte Plan - im Rahmen der zur Verfügung stehenden Zugriffsmöglichkeiten - auch effizient. Dies soll aber nicht weiter Thema dieses Artikels sein. Bei weiterem Interesse diesbezüglich verweise ich gerne auf meinen Artikel „Cost Based Optimizer Grundlagen“ aus dem Jahr 2012, der auf dieses Thema ausführlicher eingeht.

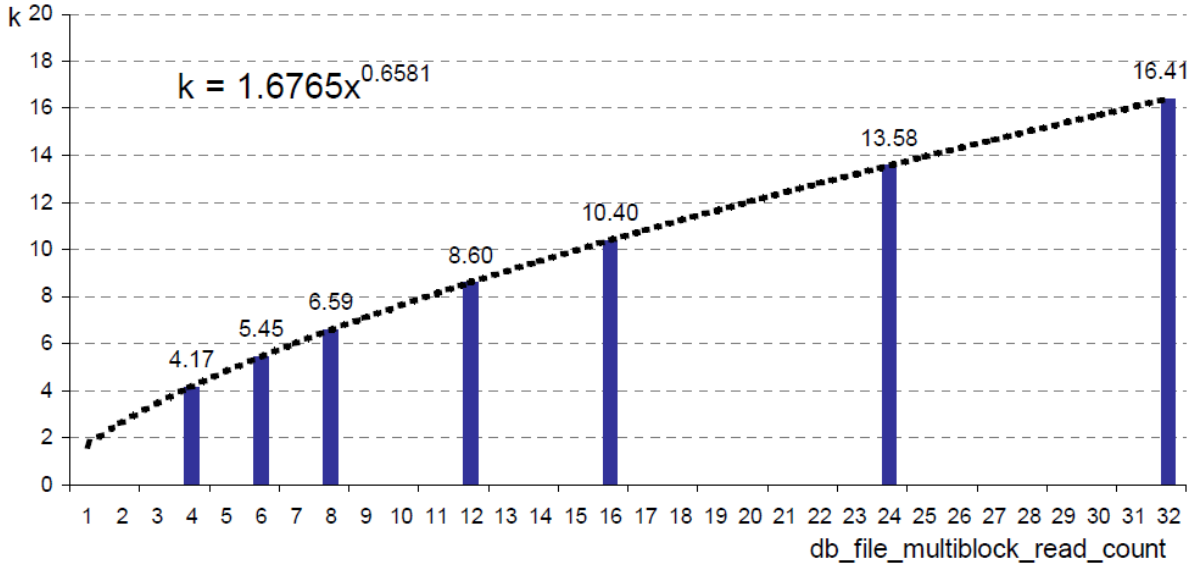
Bevor es System-Statistiken gab, verwendete der CBO für die Kostenberechnung von Tabellenzugriffen (Full Table Scans) eine interne Formel basierend auf dem eingestellten „DB_FILE_MULTIBLOCK_READ_COUNT“ Parameter, um die Kosten eines Full Table Scans im Vergleich zu einem Index-Zugriff zu berechnen.

Da diese Berechnung auch kostenmäßig keine Unterscheidung zwischen Einzelblock- und Mehrfachblockzugriffen machte – es wurde einfach die Anzahl notwendiger Zugriffe gezählt, wobei ein Einzelblockzugriff in diesem Sinne äquivalent mit einem Mehrfachlockzugriff war, und daher tendenziell Tabellenzugriffe gegenüber Indexzugriffen favorisierte - kamen mit der Version 8i noch weitere Parameter hinzu, vor allem „OPTIMIZER_INDEX_CACHING“ und „OPTIMZER_INDEX_COST_ADJ“, die auf verschiedene Art und Weise die berechneten Kosten für Index-Zugriffe beeinflussen bzw. reduzieren können.

Dieser Ansatz hatte aber verschiedene Nachteile:

1. Der eingestellte „DB_FILE_MULTIBLOCK_READ_COUNT“ ist nur sehr indirekt eine Maßzahl für die tatsächliche Performance von Full Table Scans. In der Praxis kann zum einen die beim Full Table Scan tatsächlich pro I/O-Request Anzahl gelesener Blöcke aus verschiedensten Gründen vom eingestellten Wert deutlich (mehrheitlich nach unten) abweichen, zum anderen kann der Durchsatz solcher Operationen von System zu System je nach eingestelltem Wert und eingesetzter Technologie sehr unterschiedlich sein. Die erwähnte Berechnungsformel versuchte zumindest der ersten Tatsache teilweise Rechnung zu tragen, indem der eingestellte Wert mittels einer

internen Formel entsprechend angepasst wurde – vereinfacht ausgedrückt, umso größer der Wert für „DB_FILE_MULTIBLOCK_READ_COUNT“ eingestellt, umso stärker die Reduzierung des Werts für die Kostenberechnung, Full Table Scans wurden also bei höher eingestellten Werten nicht im gleichen Maße günstiger. Zum Beispiel wurde für die Anzahl der benötigten I/O-Anfragen bei einem eingestellten Wert von 32 ein Wert von 16,41 intern stattdessen verwendet – die Anzahl der zu lesenden Blöcke also durch 16,41 anstatt 32 geteilt, um die Kosten für die Full Table Scan-Operation zu berechnen.



Tablescan cost factor k and db_file_multiblock_read_count

Abbildung 1: Anpassung der Kostenberechnung für Full Table Scans bei höheren Werten von „db_file_multiblock_read_count“ vor Verwendung von System Statistiken – Quelle: „A look under the hood of the CBO“ (2004) von Wolfgang Breitling

2. Es gab keine guten Vorgaben für die Parameter „OPTIMIZER_INDEX_CACHING“ und „OPTIMZER_INDEX_COST_ADJ“, wenn nicht die Standard-Werte verwendet werden sollen. So haben über die Jahre viele mit diesen Parametern experimentiert. Meistens entstand daraus eine Situation, in der einige SQLs davon profitiert haben und bessere Performance zeigten, während andere deutlich langsamer wurden. Insofern wurden eventuelle Probleme dadurch häufig nur verlagert, aber nicht wirklich gelöst.

3. Die beiden Parameter wurden aufgrund des oben beschriebenen Verhaltens, dass tendenziell Full Table Scans favorisiert wurden, meistens dazu eingesetzt, die Index-Kosten im Vergleich zu Full Table Scans zu reduzieren – den CBO also Index-freundlicher zu konfigurieren. Bei entsprechend extremen Einstellungen, wie bis heute bei einigen Software-Paketen, wie zum Beispiel „Siebel“, vom Hersteller immer noch als Vorgabe empfohlen, kam es häufig zu dem Problem, dass die Kostenabschätzungen für verschiedene Indizes so niedrig berechnet wurden, dass der CBO aufgrund von Rundungen keine sinnvolle Unterscheidungsmöglichkeit hatte und ineffiziente Index-Zugriffe gleich bewertete wie andere, deutlich effizientere Zugriffe. Da unter Umständen die Auswahl nur noch über den Index-Namen geschieht, wenn sich keine anderen Merkmale unterscheiden, konnte es zur Auswahl dieser ineffizienteren Index-Zugriffe kommen.

Aus all diesen Erfahrungen und Gründen hat Oracle mit der Version 9i die System Statistiken eingeführt, die diese Probleme lösen sollten. Insbesondere können über die System Statistiken die Kosten für Full Table Scans nach oben korrigiert werden, anstatt die Kosten für Index-Zugriffe zu verringern, was die erwähnten Parameter zur Anpassung der Index-Kosten überflüssig machen sollte - tatsächlich mag es Situationen geben, wo zumindest der Einsatz von angepassten „OPTIMIZER_INDEX_CACHING“-Einstellungen aufgrund der Nicht-Berücksichtigung von Caching-Effekten bei der Kostenberechnung immer noch Sinn machen kann - und das beschriebene Problem der Rundungen bei niedrigen Kosten für Index-Zugriffe umgeht - abgesehen davon, dass der CBO bei der Verwendung von System Statistiken die berechneten Kosten eben nicht mehr rundet.

In Version 9i waren die System Statistiken noch optional und wurden wahrscheinlich nur sehr begrenzt eingesetzt. Die große Umstellung geschah mit Oracle 10g – seitdem arbeitet der CBO mit sogenannten „Standard“-System Statistiken, die bei Standard-Einstellungen immer aktiv sind. Da sich aufgrund dieser Veränderung im Standard-Verhalten im Grunde alle Kostenberechnungen im Vergleich zu den Zeiten vor System-Statistiken verändert haben, gab es damals auch massive Probleme beim Upgrade von Oracle 9i nach 10g – manche Leser mögen sich noch an diese Zeit erinnern. Wenn nicht per Parameter ein 9i-Verhalten des Optimizers erzwungen wurde (OPTIMIZER_FEATURES_ENABLE), änderten sich potentiell sehr viele Ausführungspläne und aufgrund der Vielzahl an Fehlerquellen, die die Berechnungen

des CBO beeinflussen können, waren viele diese Änderungen nicht positiv, es kam also bei vielen Kunden zu Performanceproblemen nach dem Upgrade.
Diese „Standard“-System Statistiken sind seitdem bis heute aktiv und können nur über undokumentierte Parameter, oder einem Zurückstellen des Verhaltens per OPTIMIZER_FEATURES_ENABLE deaktiviert werden. Dies bedeutet: System Statistiken sind grundsätzlich relevant, auch wenn man keine aktiven Schritte unternommen hat, um diese zu verwenden.

Wie System Statistiken funktionieren

Die System Statistiken umfassen verschiedene Messwerte, die die Fähigkeiten von CPU und I/O beschreiben. Diese erlauben dem CBO, verschiedene Berechnungen anders als zuvor durchzuführen:

- Für Einzelblockzugriffe und Mehrfachblockzugriffe beschreiben die System Statistiken, wie lange diese durchschnittlich benötigen (SREADTIM und MREADTIM genannt), also eine Zeitangabe in Millisekunden. Dies löst als Seiteneffekt oben beschriebenes Problem, da jetzt die beiden Zugriffsarten unterschiedlich bewertet werden im Sinne der berechneten Kosten – die Annahme ist, dass ein Mehrfachblockzugriff länger als ein Einzelblockzugriff dauert – was in der Realität nicht immer unbedingt der Fall sein muss, da es zum Beispiel SAN-Systeme mit Read-Ahead-Effekten gibt, bei denen Daten für Full Table Scans schon im SAN-Cache vorliegen und daher sehr schnell gelesen werden können. Die Zeitangaben erlauben dem CBO auch, die geschätzten Kosten auch in Zeit auszudrücken – dies ist in der Spalte „TIME“ eines Ausführungsplans bei Verwendung von System Statistiken zu finden. Intern wird der Berechnungsmodus unter Verwendung von System Statistiken auch „CPU Costing“ genannt, warum, siehe nächster Punkt
- Die CPU-Geschwindigkeit wird auch gemessen (CPUSPEED bzw. CPUSPEEDNW), und eine CPU-Kosten- bzw. Zeitkomponente berechnet, basierend auf den durchgeführten Operationen. Jeder Zeilen- bzw. Spaltenzugriff benötigt CPU-Zeit, je nach Position der Spalte in einer Zeile auch unterschiedlich. Angewendete SQL-Funktionen (zum Beispiel UPPER, LOWER, TO_CHAR etc.) werden ebenso im Sinne von CPU-Zeit berücksichtigt. Dies erlaubt dem CBO auch, Filter in einer intelligenten Art und Weise anzuwenden (Predicate re-ordering) – indem diese zuerst ausgewertet

werden, die weniger CPU verbrauchen und stärker filtern und erst danach andere Prädikate, die CPU-intensiver sind

- Ein weiterer, maßgeblicher Messwert beschreibt, wie viele Blöcke Oracle tatsächlich durchschnittlich bei einem Mehrfachblockzugriff lesen kann (MBRC). Ist dieser Messwert verfügbar, ergibt sich eine Unabhängigkeit der Kostenberechnung von einem eventuell eingestellten „DB_FILE_MULTIBLOCK_READ_COUNT“-Parameter, und es wird dieser gemessene Wert stattdessen verwendet. Je nach Konfiguration (Parameter explizit gesetzt oder Standardwert) und Art und Weise der Messung hat der „DB_FILE_MULTIBLOCK_READ_COUNT“-Parameter dann doch wieder indirekt einen Einfluss, da bei explizit gesetztem Parameter Oracle diesen in den meisten Fällen als Obergrenze für die Anzahl zu lesender Blöcke pro Operation verwendet
- Weitere Messwerte beziehen sich auf Parallelverarbeitung (Parallel Execution) und geben an, was als durchschnittlicher I/O-Durchsatz eines einzelnen Parallel Execution Servers gemessen wurde, und was der maximale Durchsatz des gesamten Systems war. Später dazu mehr, wie diese Parameter die Kostenberechnung beeinflussen können

Wie werden System Statistiken konfiguriert?

Wichtig: Nicht immer sind all diese Messwerte verfügbar. Recht verwirrend bei System Statistiken ist die Tatsache, dass es unterschiedliche Arten gibt, je nachdem, auf welche Art und Weise diese ermittelt wurden (WORKLOAD und NOWORKLOAD).

Für die Konfiguration von System Statistiken steht ein umfangreiches Interface im DBMS_STATS-Paket zur Verfügung. Mittels dieses Interfaces können diese per Messung ermittelt, exportiert, importiert, ausgelesen, gelöscht und auch manuell gesetzt werden.

Die sogenannten NOWORKLOAD System Statistiken, zu denen auch die mit 10g eingeführten Standard-System Statistiken gehören, können per entsprechendem Aufruf im DBMS_STATS-Paket (GATHER_SYSTEM_STATS) gemessen werden.

Sie ermitteln nur einen eingeschränkten Umfang an Werten:

CPU-Geschwindigkeit in Oracles eigener Einheit (CPUSPEEDNW)

Durchschnittliche Zeit zur Festplatten-Kopfpositionierung in Millisekunden (IOSEEKTIM)
Durchschnittliche I/O-Datenübertragungsrate von Festplatte in Bytes pro Millisekunde (IOTFRSPEED)
Durchschnittliche Anzahl Blöcke bei Mehrfachblockzugriff (MBRC) – nur im neuen EXADATA-Modus, später dazu mehr

Die Messung von NOWORKLOAD-System Statistiken erfolgt mittels einer künstlichen Last, die die Datenbank bei der Messung selbst erzeugt – daher der Name NOWORKLOAD, da die Messung nicht auf einem Lastprofil, die eine Applikation verursacht, basiert. Idealerweise sollte so eine Messung daher auch auf einem System mit möglichst wenig anderer Last durchgeführt werden, also am besten „im Leerlauf“.

Im Gegensatz dazu basieren die sogenannten WORKLOAD-System Statistiken auf der Messung einer tatsächlichen Last auf der Datenbank. Hier wird also von der Datenbank während der Messung (auch mittels GATHER_SYSTEM_STATS) keine künstliche Last erzeugt, sondern die tatsächliche Aktivität gemessen. Das heißt auch, dass die Datenbank während einer solchen Messung aktiv sein muss, also ohne eine möglichst repräsentative, durch eine Anwendung erzeugte Last so eine Messung keinen Sinn macht. Dabei werden umfangreichere Messwerte ermittelt:
CPU-Geschwindigkeit in Oracles eigener Einheit (CPUSPEED)
Durchschnittliche Dauer Einzelblockzugriff in Millisekunden (SREADTIM)
Durchschnittliche Dauer Mehrfachblockzugriff in Millisekunden (MREADTIM)
Durchschnittliche Anzahl Blöcke bei Mehrfachblockzugriff (MBRC)
Maximaler I/O-Durchsatz des Systems in Bytes pro Sekunde (MAXTHR)
Durchschnittlicher I/O-Durchsatz eines Parallel Execution Servers in Bytes pro Sekunde (SLAVETHR)

Bei der Messung von WORKLOAD System Statistiken hängt die Verfügbarkeit mancher Messwerte davon ab, welche Operationen bei den Messungen tatsächlich durchgeführt wurden – wurden zum Beispiel keine Mehrfachblockzugriffe während einer entsprechenden Messung durchgeführt, wird sowohl der oben erwähnte MBRC als auch MREADTIM nicht in den Systemstatistiken verfügbar sein. Je nachdem,

welche Messwerte fehlen oder auch bestimmten Überprüfungen nicht standhalten (wie zum Beispiel, dass MREADTIM größer SREADTIM sein muss, siehe oben, dass das nicht immer in der Realität der Fall sein muss), werden diese entweder mit Standard/Ersatz-Werten ersetzt oder die Art der Kostenberechnung fällt auf eine andere Art der Berechnung zurück (NOWORKLOAD anstatt von WORKLOAD).

Die Standard System-Statistiken umfassen nur einen sehr eingeschränkten Teil dieser Messwerte und gehören zu der NOWORKLOAD-Art. Dadurch definieren sie nur die CPU-Geschwindigkeit (CPUSPEEDNW) und mittels einer indirekten Berechnung die Zeiten für Einzelblockzugriffe und Mehrfachblockzugriffe (SREADTIM und MREADTIM, berechnet mittels Blockgröße, IOSEEKTIM und IOTFRSPEED).

```
Default System Statistics values:
IOSEEKTIM = 10 ms
IOTFRSPEED = 4096 bytes per millisecond = approx. 4 MB per second
db_block_size = 8192 = 8 KB
mbrc = 8 (default used for cost calculation when
“db_file_multiblock_read_count” is left unset)
SREADTIM = IOSEEKTIM + db_block_size / IOTFRSPEED

MREADTIM = IOSEEKTIM + mbrc * db_block_size / IOTFRSPEED

SREADTIM = 10 + 8192 / 4096 = 10 + 2 = 12ms

MREADTIM = 10 + 8 * 8192 / 4096 = 10 + 16 = 26ms
```

Abbildung 2: Beispielhafte Berechnung/Ableitung der SREADTIM und MREADTIM-Werte bei Verwendung der Standard NOWORKLOAD-System Statistiken, einer Standard-Blockgröße von 8 KB und ungesetztem „DB_FILE_MULTIBLOCK_READ_COUNT“ (wie von Oracle seit Version 10g empfohlen)

Wer mehr zu den Details der verschiedenen Arten und die dazugehörigen Berechnungen erfahren möchte, kann ich meine (schon recht alte) Blogartikel-Serie dazu empfehlen (https://oracle-randolf.blogspot.com/2009/04/understanding-different-modes-of-system.html), und vor allem Christian Antogninis Buch „Troubleshooting Oracle Performance“, das sogar ein eigenes Kapitel den System Statistiken widmet und auch aktuellere Entwicklungen berücksichtigt (zumindest inklusive Version Oracle 12.1).

Gibt es Empfehlungen / Best Practices bezüglich System Statistiken?

Eine der Probleme im Umgang mit System Statistiken ist, dass es recht unterschiedliche Aussagen gibt. Als diese eingeführt wurden, gab es offizielle Empfehlungen seitens Oracle, diese explizit zu messen, und zwar in Form von WORKLOAD-System Statistiken, also gemessen basierend auf dem tatsächlichen Last-Profil der Datenbank (siehe zum Beispiel https://docs.oracle.com/cd/A91202_01/901_doc/server.901/a87503/stats.htm#28811) . Christian Antognini stuft System Statistiken als Initialisierungsparameter in seinem Buch „Troubleshooting Oracle Performance“ ein und empfiehlt weiterhin grundsätzlich, diese als WORKLOAD System Statistiken zu ermitteln, geht allerdings detaillierter auf den Prozess ein und stellt verschiedene Alternativen vor.

Die offizielle Sprachregelung seitens Oracle hat sich in der Zwischenzeit geändert – dort empfiehlt man inzwischen, nur noch mit den Standardwerten zu arbeiten, und die gemessenen System Statistiken nur dann zu verwenden, wenn man entweder bereits positive Erfahrungen damit gemacht hat, also eine bereits etablierte Umgebung mit entsprechenden System Statistiken hat, oder an einem Punkt ist, an dem man die Auswirkungen ausführlich testen kann und wird (zum Beispiel neue Hardware, Upgrade etc.) und dazu bereit ist, den Mehraufwand des Managements von System Statistiken in Kauf zu nehmen (siehe zum Beispiel https://sqlmaria.com/2018/04/10/should-you-gather-system-statistics/ und https://blogs.oracle.com/optimizer/should-you-gather-system-statistics)

Eine der Probleme mit dem Messen von System Statistiken ist, dass die Messwerte sich deutlich unterscheiden können – die Resultate sind also keineswegs konsistent, sondern können stark schwanken je nach Messzeitpunkt und Aktivität in der Datenbank. Von daher kann man zumindest die Empfehlung aussprechen, System Statistiken **nicht** regelmäßig zu aktualisieren, wie ich es schon bei einigen Kunden vorgefunden habe, die zum Beispiel einen wöchentlichen oder täglichen Job zur Ermittlung etabliert hatten. Aufgrund der Tatsache, dass System Statistiken direkte Auswirkungen auf alle Ausführungspläne haben können, führt man auf diese Art und Weise eine potentielle Instabilität in die Umgebung ein, auf die man möglichst verzichten sollte.

Eine Idee, die Christian Antognini auch als mögliche Alternative empfiehlt, ist die Variante, die System Statistiken regelmäßig zu ermitteln, aber nicht direkt zu aktivieren, sondern in einer entsprechenden Backup-Tabelle zu speichern. Diese

Option ist vielen nicht bekannt, wird aber von den meisten DBMS_STATS-Aufrufen unterstützt. Mittels dieser Methode kann man die ermittelten Messwerte auch auf ihre Volatilität hin überprüfen und dann entscheiden, welche Werte am ehesten Sinn machen könnten.
Egal für welche Variante man sich entscheidet, klar sollte sein, dass System Statistiken kein Allheilmittel sind, und es meistens eine Gruppe von Abfragen geben wird, die weiterhin nicht optimal performen und um die man sich getrennt kümmern muss. Zwar sollten System Statistiken in der Theorie dem CBO helfen, auf die jeweilige Umgebung angepasst die effizientesten Ausführungspläne zu finden. In der Praxis ist dies jedoch nicht immer der Fall und der Tatsache geschuldet, dass es viele Parameter gibt, die die Entscheidungen des CBOs beeinflussen (System Statistiken, Objekt Statistiken, Art der Abfragen und Filter, eventuell Dynamic Sampling / Statistics etc.) und insofern die System Statistiken nur einen Teil davon darstellen. Je nachdem, wie weit die anderen abgeschätzten Eingabewerte für die Kostenberechnung von der Realität abweichen, können hier auch Effekte in der Art auftreten, dass eigentlich besser auf die Umgebung abgestimmte System Statistiken genau den gegenteiligen Effekt erreichen und es im Endeffekt / Summe zu weniger effizienten Ausführungsplänen kommt.

Wie sieht der praktische Nutzen von System Statistiken aus?
Neben der Berücksichtigung der CPU-Komponente ist also die unterschiedliche und konfigurierbare Bewertung von Einzel- und Mehrfachblockzugriffen die wichtigste Neuerung, die mit System Statistiken eingeführt wurde. Der Hintergedanke bei der Einführung der Standard-System Statistiken in Oracle 10g war offensichtlich, die Kostenberechnung des CBOs Index-freundlicher zu gestalten – dieses Ziel wurde grundsätzlich erreicht – so steigen die Kosten für einen Full Table Scan je nach eingestelltem / gemessenen MBRC (Durchschnittliche Anzahl Blöcke bei Mehrfachblockzugriff) zwischen Faktor 1,8 und 7 im Vergleich zur Kostenberechnung ohne System Statistiken – mehr Details dazu finden sich in meiner oben verlinkten Blog-Serie.

```
Full Table Scan costs without System Statistics:
-----------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost  |
-----------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1518 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |       |
```

```
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1518 |
-------------------------------------------------------
Full Table Scan costs with default System Statistics:
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2709   (0)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2709   (0)| 00:00:33 |
-------------------------------------------------------------------------------
```

Abbildung 3: Beispielhafte Kostenberechnung für einen Full Table Scan mit und ohne System Statistiken

Wichtig in diesem Zusammenhang ist zu verstehen, dass es neben der angenommenen Anzahl Blöcke pro Mehrfachblockzugriff (MBRC) maßgeblich das Verhältnis zwischen angenommener Zeit für Einzelblock- und Mehrfachblockzugriff (SREADTIM und MREADTIM) ist, das für die Kostenberechnung relevant ist, nicht so sehr die absoluten Werte.

Möchte man also die Kostenberechnung Index-freundlicher gestalten, kann man dies über das Verhältnis von SREADTIM und MREADTIM beeinflussen. Ebenso wird die Kostenberechnung Index-freundlicher, wenn der MBRC (Durchschnittliche Anzahl Blöcke bei Mehrfachblockzugriff) niedriger eingestellt wird – auch ein manuelles Setzen von System Statistiken per DBMS_STATS.SET_SYSTEM_STATS ist möglich.

Umgekehrt kann man Full Table Scans favorisieren, indem das Verhältnis zwischen MREADTIM und SREADTIM kleiner und/oder der MBRC größer eingestellt wird.

All dies kann im Grunde auch mit dem Parameter OPTIMIZER_INDEX_COST_ADJ erreicht werden – allerdings gibt es zwei wesentliche Unterschiede:

1. Wenn der CBO mittels des Parameters OPTIMIZER_INDEX_COST_ADJ Index-freundlicher konfiguriert werden soll, verringern sich die Kosten für Index-Zugriffe. Bei der Anpassung mittels System Statistiken erhöhen sich die Kosten für Full Table Scans – die Kosten für den Index-Zugriff bleiben unverändert
2. Während es keine offiziell dokumentierte Möglichkeit gibt, einen passenden Wert für OPTIMIZER_INDEX_COST_ADJ per Messung oder Statistiken festzulegen, können System Statistiken über ein offiziell dokumentiertes Interface gemessen werden und repräsentieren somit in der Theorie die CPU- und I/O-Möglichkeiten der jeweiligen Umgebung.

Aufgrund der weiter oben beschriebenen Effekte der möglichen Volatilität bei der Messung der System Statistiken, dem Zusammenspiel mit anderen Eingabewerten bei der Kostenberechnung (vor allem Mengenabschätzungen und Clustering Factor von Indizes) und auch der Tatsache, dass das Kostenmodell des CBOs grundsätzlich von physischem I/O ausgeht und Caching nicht berücksichtigt, müssen System Statistiken in der Realität nicht unbedingt zu den erwarteten Verbesserungen bei der Kostenberechnung führen. Gerade Ausführungspläne, die von Caching stark profitieren, können weiterhin falsch vom CBO eingeschätzt werden. Daran ändern auch System Statistiken nichts

Handelt es sich um eine Umgebung, die auch die Parallelverarbeitungsoption der Enterprise Edition einsetzt (Parallel Execution Features), können System Statistiken auch dazu eingesetzt werden, die Kostenberechnung für parallele Ausführungspläne zu beeinflussen. Insbesondere können über die Parameter „SLAVETHR“ und „MAXTHR“ zu optimistische Kostenberechnungen nach unten korrigiert werden. Ohne diese Parameter nimmt der CBO zum einen an, dass die Parallelverarbeitung im Sinne der Kostenberechnung unendlich skaliert (was auf jeden Fall unrealistisch ist) und zum anderen, dass die Parallelverarbeitung ungefähr mit dem Faktor 0.9 der seriellen Verarbeitung skaliert. Mittels „SLAVETHR“ kann letzterer Faktor nach unten korrigiert werden – dem CBO also mitgeteilt werden, dass der Faktor niedriger als 0.9 liegt, die Parallelverarbeitung also schlechter skaliert. Eine bessere Skalierbarkeit als 0.9 kann damit nicht erreicht werden, man kann also den Maximalwert von 0.9 nicht überschreiten – ansonsten wird „SLAVETHR“ ignoriert und 0.9 angenommen. Damit können die Kostenberechnungen für Parallelverarbeitungen höher eingestellt werden – der CBO wird also „parallelunfreundlicher“ konfiguriert und eventuell verfügbare, serielle Index-basierte Zugriffswege potentiell favorisiert.

```
select max(n1) from t1;

-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   278   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   278   (0)| 00:00:02 |
```

```
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   278   (0)| 00:00:02 |
-------------------------------------------------------------------------------

Increased parallel cost at same degree when SLAVETHR is lower than default value,
which is 0.9 * MBRC * db_block_size / MREADTIM bytes per millisecond

-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   800   (0)| 00:00:05 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   800   (0)| 00:00:05 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   800   (0)| 00:00:05 |
-------------------------------------------------------------------------------
```

Abbildung 4: Höhere Kosten bei der Berechnung von Parallel Execution Plänen und der Verwendung von SLAVETHR-Werten unterhalb des Standard-Wertes

Mittels „MAXTHR“ kann der standardmäßigen, „unendlichen“ Skalierbarkeit eine Obergrenze gesetzt werden – ab einem bestimmten Grad an Parallelität verringern sich dann die berechneten Kosten nicht mehr, was durchaus Sinn macht und insbesondere hilft zu verhindern, dass unrealistisch hohe Parallelitätsgrade den CBO einen Ausführungsplan bevorzugen lassen, der in der Realität so nicht skalieren kann. Allerdings gilt dies nur für die Kostenberechnung – wenn nicht andere Restriktionen über Parameter oder den Resource Manager eingreifen, würde bei der Ausführung eines eventuell dann ausgewählten Ausführungsplans mit hohem Parallelitätsgrad diese dann doch zur Laufzeit angewendet werden.

```
select max(n1) from t1;

Serial cost:
-----------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     5 |  1280   (3)| 00:00:07 |
|   1 |  SORT AGGREGATE    |      |     1 |     5 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 40000 |   195K|  1280   (3)| 00:00:07 |
-----------------------------------------------------------------------

Parallel(T1 42) cost without MAXTHR set
```

```
-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |    33   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|    33   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|    33   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Parallel(T1 42) cost with MAXTHR set
-------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |     1 |     5 |   250   (0)| 00:00:02 |
|   1 |  SORT AGGREGATE        |          |     1 |     5 |            |          |
|   2 |   PX COORDINATOR       |          |       |       |            |          |
|   3 |    PX SEND QC (RANDOM) | :TQ10000 |     1 |     5 |            |          |
|   4 |     SORT AGGREGATE     |          |     1 |     5 |            |          |
|   5 |      PX BLOCK ITERATOR |          | 40000 |   195K|   250   (0)| 00:00:02 |
|   6 |       TABLE ACCESS FULL| T1       | 40000 |   195K|   250   (0)| 00:00:02 |
-------------------------------------------------------------------------------
```

Abbildung 5: Berücksichtigung der limitierten Skalierbarkeit von Parallel Execution bei der Kostenberechnung durch MAXTHR

Neuerungen bezüglich System Statistiken

In Bezug auf System Statistiken haben sich in den letzten Jahren nur wenige Veränderungen ergeben. Die meisten Änderungen haben sich auf Bugs in der Berechnung bezogen. So wurde mit der Version 11.2.0.1 und 11.2.0.2 ein Fehler eingeführt, der völlig unrealistische Werte für die Parameter SREADTIM und MREADTIM bei der Messung von WORKLOAD System Statistiken berechnet hat, und bei der Berechnung der parallelen Kosten, die SLAVETHR und MAXTHR verwenden, wurde auch ein fragwürdiger Berechnungsfaktor (und die verwendete Einheit) verändert. Alle diese Fehler sind ab der Version 11.2.0.3 behoben.
Mit der Version 12c (und auch verfügbar ab Version 11.2.0.4) wurden sogenannte „EXADATA“ System Statistiken eingeführt. Hier handelt es sich um eine Abwandlung der sogenannten NOWORKLOAD System Statistiken. Die Einführung dieses neuen Modus hat verschiedene Gründe:

- Der CBO berücksichtigt bei seinen Kostenberechnungen standardmäßig nicht den möglichen Zeitgewinn der Exadata Smart Scans – im Gegensatz zu In-

Memory Scans des In-Memory Column Stores, die der CBO explizit entsprechend günstiger bewertet

- Die Messungen von System Statistiken funktionieren erfahrungsgemäß auf Exadata-Systemen nicht wirklich gut. Das hat mit technischen Details zu tun, wie gemessen wird und wie Smart Scans auf Exadata funktionieren (sogenannte Buffer Cache Reads vs. Direct Path Reads)

Der EXADATA Modus entspricht im Grunde einer Messung der NOWORKLOAD System Statistiken, setzt aber zusätzlich den Parameter MBRC (Durchschnittliche Anzahl Blöcke bei Mehrfachblockzugriff) auf den Wert von DB_FILE_MULTIBLOCK_READ_COUNT. Dies verändert die Kostenberechnung also tatsächlich nur dann, wenn der Parameter DB_FILE_MULTIBLOCK_READ_COUNT nicht explizit gesetzt ist, da ansonsten für die Berechnung schon der explizit eingestellte Wert benutzt wurde. Den Parameter nicht explizit zu setzen, ist zwar seit Oracle 10g die offizielle Empfehlung, trotzdem gibt es wohl immer noch viele Installationen, die den Parameter, meistens auch aus historischen Gründen, angeben. Wird der Parameter nicht explizit gesetzt, verwendet Oracle zwei verschiedene Werte für die Kostenberechnung von Full Table Scans und bei der tatsächlichen Ausführung. Die Kostenberechnung verwendet den Wert 8, was relativ wenig ist und daher Full Table Scans relativ teuer macht – es war ja die Intention bei der Einführung von System Statistiken, die Kostenberechnung standardmäßig Index-freundlicher zu gestalten – während bei der Ausführung ein deutlich höherer Wert zum Einsatz kommt. Wenn das Verhältnis zwischen SGA-Größe und maximaler Anzahl konfigurierter Prozesse nicht ein bestimmtes Verhältnis unterschreitet, also nicht sehr viele Prozesse für eine recht kleine SGA konfiguriert sind, dann wird bei der Ausführung versucht, typischerweise 1 MB Mehrfachblockzugriffe zu verwenden, was bei einer Blockgröße von 8 KB dem Wert 128 entspricht. Dieser Wert von 1 MB / 128 wird dann also im EXADATA-Modus der System Statistiken in den Wert MBRC übernommen, was die Kostenberechnung entsprechend Full Table Scan-freundlicher macht. Damit sollten EXADATA und auch andere Systeme mit sehr schnellen Full Table Scans diese in den Ausführungsplänen eher favorisieren. Sinn macht das allerdings nur dann wirklich, wenn man eine Applikation benutzt, die entsprechend ausgelegt ist, also typischerweise eine Data Warehouse-Anwendung. Da inzwischen auch viele OLTP-Anwendungen oder Mixturen aus beiden auf solchen Plattformen betrieben werden,

mag eine solche Favorisierung von Full Table Scans für diese Art der Verwendung nicht unbedingt hilfreich sein.

```
Sample Full Table Cost with default System Statistics and
"db_file_multiblock_read_count" left unset
 ------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  2716   (1)| 00:00:33 |
------------------------------------------------------------------------------

Sample Full Table Cost with EXADATA System Statistics, which sets MBRC to 128 in
most cases when using 8 KB default block size and leaving
"db_file_multiblock_read_count" unset
------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |     4 |  1738   (1)| 00:00:21 |
|   1 |  SORT AGGREGATE    |      |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| T1   | 10000 | 40000 |  1738   (1)| 00:00:21 |
------------------------------------------------------------------------------
```

Abbildung 6: Verringerte Kosten für einen Full Table Scan bei Verwendung von EXADATA-System Statistiken

Historie der I/O Kalibrierung

Zusätzlich zu den mit Oracle 9i eingeführten System Statistiken kam zusammen mit dem neuen „Auto DOP“-Feature in Oracle 11.2 die sogenannte I/O-Kalibrierung. Das „Auto DOP“-Feature dreht sich hauptsächlich um die automatische Berechnung des Parallelisierungsgrades bei Parallelausführungen. Diese Berechnung wiederum fußt maßgeblich auf dem mittels I/O Kalibrierung ermittelten I/O Durchsatz. Auf den ersten Blick erscheint es allerdings merkwürdig, warum diese I/O Kalibrierung zusätzlich zu den bereits bestehenden System Statistiken eingeführt wurde – geht es doch bei beiden maßgeblich um die Messung von I/O Durchsätzen – auch wenn sowohl die System Statistiken als auch die I/O Kalibrierung noch andere Werte umfassen.

Der Hintergrund dieser Überschneidung kommt wahrscheinlich daher, dass die System Statistiken nicht unbedingt gemessen werden müssen, bzw. eine Messung dieser alle Ausführungspläne beeinflussen können. Anstatt also das potentielle Risiko einzugehen, Kunden, die das „Auto DOP“-Feature verwenden wollen, dazu zu

zwingen, die System Statistiken zu ermitteln und damit eine globale Auswirkungen auf Ausführungspläne zu verursachen, hat man sich wohl für diese separate Messung entschieden, die dann nur für das „Auto DOP“-Feature relevant ist.
Wer genau hinschaut, wird merken, dass sich die Spalte „TIME“, also die Zeitabschätzung des CBOs im Rahmen der Ausführungsplanerstellung, die vor der I/O Kalibrierung eine direkte Relation zu den Kosten hatte (Kosten mal Zeit für einen Einzelblockzugriff (SREADTIM) gleich Zeit), mit Vorhandensein von I/O Kalibrierungswerten entsprechend verändert – und damit direkten Einfluss auf die Entscheidung hat, ob bei Verwendung von „Auto DOP“ eine Ausführung überhaupt parallelisiert wird – der Parameter PARALLEL_MIN_TIME_THRESHOLD steht standardmäßig auf 10 Sekunden. Nur wenn also gemäß der neuen TIME-Berechnung eine Ausführung geschätzt länger als dieser Parameterwert dauert, wird Parallelverarbeitung in Betracht gezogen.

```
TIME represents cost multiplied by SREADTIM without I/O calibration, in this case
here 2716 * 12 ms equals approx. 32.6 seconds
-------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:33 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:33 |
-------------------------------------------------------------------
New calculation of TIME in the presence of I/O calibration results
-------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |  2716   (1)| 00:00:06 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T    | 10000 |  2716   (1)| 00:00:06 |
-------------------------------------------------------------------
```

Abbildung 7: Veränderte Zeit (TIME)-Berechnung bei Vorhandensein von I/O-Kalibrierungswerten

Wie funktioniert die I/O Kalibrierung?

Leider ist das Handling der I/O-Kalibrierung bei weitem nicht so ausgereift, wie bei den System Statistiken, die über das DBMS_STATS-Paket eine umfassende API besitzen.
Stattdessen gibt es genau einen Aufruf im Paket DBMS_RESOURCE_MANAGER, das die I/O-Kalibrierung durchführt, genannt CALIBRATE_IO. Dessen Parameter

sind nicht wirklich gut dokumentiert – man soll die Anzahl der physischen Festplatten angeben, die der Datenbank zur Verfügung stehen und die „maximale Latenz“ für einen Festplattenzugriff. Was der Sinn vor allem des letzteren Parameters sein soll, ist nicht weiter dokumentiert. Vorausgesetzt, asynchrones I/O ist auf allen Data Files aktiviert – andernfalls gibt es eine Fehlermeldung – führt ein Aufruf der Prozedur einen künstlichen Benchmark aus und sollte also daher zu einem Zeitpunkt durchgeführt werden, zu dem möglichst wenig Aktivität auf dem System ist. Dieser künstliche Benchmark erzeugt sowohl Einzelblock- als auch Mehrfachblockzugriffe und ermittelt darüber eine maximale IOPS-Rate, den maximalen I/O-Durchsatz als auch eine Latenz. Der Messvorgang kann in V$IO_CALIBRATION_STATUS überwacht und beobachtet werden – die Ergebnisse nach Abschluss stehen dann in DBA_RSRC_IO_CALIBRATE zur Verfügung.

Interessanterweise ist bisher nur einer dieser Messwerte relevant für die Berechnung des „Auto DOP“ – der sogenannte MAX_PMBPS, was dem I/O-Durchsatz in Megabyte pro Sekunde eines Parallel Execution Servers entspricht – und damit eigentlich genau dem Messwert SLAVETHR der System Statistiken (auch wenn die Einheit unterschiedlich ist, Bytes pro Sekunde vs. Megabyte pro Sekunde).

Leider hat die I/O Kalibrierung in der Praxis ähnliche Probleme wie die Messung der System Statistiken – die Ergebnisse können sehr stark schwanken, und vor allem auf Systemen mit sehr schnellem I/O – zum Beispiel EXADATA – nicht wirklich repräsentative Werte ermitteln.

Zusätzlich ist wichtig zu verstehen, dass Änderungen an den I/O Kalibrierung nur nach Neustart der Instanz gültig werden – im Gegensatz zu System Statistiken, die jederzeit online verändert werden können und sich sofort auswirken.

Ein weiterer signifikanter Unterschied ist die fehlende API zur Pflege der I/O Kalibrierungswerte – so empfiehlt Oracle in Versionen vor 12c im Falle von ungünstig ermittelten Werten, oder generell auf EXADATA-Systemen bzw. Systemen mit schnellem I/O, die I/O Kalibrierungswerte manuell zu setzen.

Allerdings ist dieses manuelle Setzen im Gegensatz zu den System Statistiken, bei denen Werte direkt mittels entsprechender DBMS_STATS.SET_SYSTEM_STATS-Aufrufe gesetzt werden können, nur durch die direkte Manipulation einer SYS-Tabelle (SYS.RESOURCE_IO_CALIBRATE$, die Tabelle hinter dem View DBA_RSRC_IO_CALIBRATE) möglich – ein eher ungewöhnliches Vorgehen, das zeigt, dass eine entsprechende, offizielle API hilfreich wäre.

Neuerungen bezüglich der I/O Kalibrierung

Ähnlich wie bei den System Statistiken und Oracle 10g hat Oracle mit der Version 12c sogenannte Standardwerte für die I/O Kalibrierung eingeführt. In Versionen vor 12c mussten diese Werte vorhanden sein – entweder echt gemessen oder eben manuell per DELETE/INSERT gesetzt, um die Auto DOP-Berechnung verwenden zu können. Ansonsten wurde eine entsprechende Anmerkung bei der Planerstellung generiert, die darauf hingewiesen hat, dass die Werte fehlen, und der Parallelitätsgrad wurde über die bisherigen („Manual DOP“) Verfahren festgelegt.
Ab Oracle 12c wird bei nicht vorhandenen Werten einfach der bisher für EXADATA-Systeme empfohlene Wert von 200 MB pro Sekunde für den Messwert MAX_PMBPS angenommen und die Berechnung entsprechend durchgeführt.
Zusätzlich – damit man nicht an diesen Wert manipulieren muss, wurde ein neuer Parameter PARALLEL_DEGREE_LEVEL eingeführt, mit dessen Hilfe man die „Auto DOP“-Berechnung entsprechend verändern kann. Der Standardwert beträgt 100, Werte kleiner als 100 ergeben niedrigere Parallelitätsgrade als normal, entsprechend höhere Werte ergeben höhere Grade.
Ein Seiteneffekt der neuen Standardwerte für die I/O Kalibrierung ist, dass SQLs, die einen PARALLEL-Hint auf Statement-Ebene beinhalten – ohne weitere Angabe – also einfach SELECT /*+ PARALLEL */ …, ab Version 12c auch ohne I/O-Kalibrierungswerte die „Auto DOP“-Berechnung aktivieren, während in älteren 11.2 Versionen die fehlenden Werte eben zu dem Rückfall auf „Manual DOP“ führen. Eine signifikante Änderung im Verhalten, die bei Entwicklern, die sich über die genaue Auswirkung des Hints nicht klar waren, zu Überraschungen bei einem Upgrade führen können.

```
Change in behaviour between 11.2 and 12c/18c when using PARALLEL statement level
hint

select /*+ parallel */ * from t2 in 11.2.0.4 without I/O calibration results,
fallback to default parallel degree, which is 8 in this case here:
------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|   651   (1)| 00:00:08 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|   651   (1)| 00:00:08 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|   651   (1)| 00:00:08 |
```

```
-------------------------------------------------------------------------------

Note
-----
   - automatic DOP: skipped because of IO calibrate statistics are missing


select /*+ parallel */ * from t2 in 12.2.0.1 without using I/O calibration results,
using default values now, computes a degree of 2 here:
-------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   1 |  PX COORDINATOR      |          |       |       |            |          |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  1000K|   113M|  2600   (1)| 00:00:01 |
|   3 |    PX BLOCK ITERATOR |          |  1000K|   113M|  2600   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| T2       |  1000K|   113M|  2600   (1)| 00:00:01 |
-------------------------------------------------------------------------------

Automatic Degree of Parallelism Information:
-------------------------------------------

   - Degree of Parallelism of 2 is derived from scan of object CBO_TEST.T2_11204

Note
-----
   - automatic DOP: Computed Degree of Parallelism is 2
```

Abbildung 8: Verändertes Verhalten bei fehlenden I/O Kalibrierungsergebnissen und Verwendung des PARALLEL-Hints auf Statement-Ebene ab Oracle 12c durch Verwendung/Annahme von Standard-Werten für die I/O Kalibrierung

Kontakt:

Randolf Geist

http://www.oracle-performance.de

randolf.geist@oracle-performance.de