Oracle Optimizer System-Statistiken-Update 2018

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. Zudem 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. Sie werden vom Oracle Optimizer dazu verwendet, die Kosten für Index- und Tabellen-Zugriffe zu berechnen.

Historie der System-Statistiken

Seit Einführung des kostenbasierten Optimizer (Cost Based Optimizer, CBO) in der Version 7 geschieht die Auswahl des auszuführenden Plans, der 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, in Form der berechneten Kosten; 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 CBO 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. Die Einheit der berechneten Kosten ist der Einzelblock-Zugriff („Single Block Read"), also der Zugriff auf einen einzelnen Block. Insofern stellt der CBO eine Art Compiler dar, der SQL in einen ausführbaren Pseudocode übersetzt. Dabei entsteht eine Menge möglicher Code-Alternativen, die über die Kostenabschätzung bewertet werden.

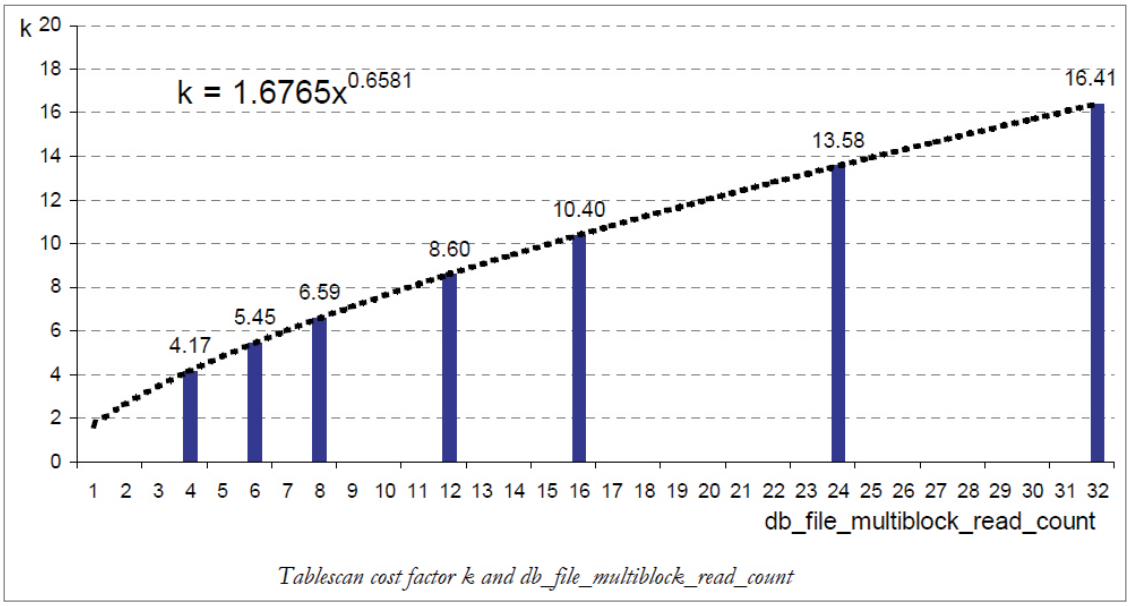


Abbildung 1: Anpassung der Kostenberechnung für Full Table Scans bei höheren Werten von „db_file_multiblock_read_count" vor Verwendung von System-Statistiken, Quelle: „A look under the hood of the CBO" (2004) von Wolfgang Breitling

Wesentliche Anteile der berechneten Kosten beruhen auf Mengen-Abschätzungen und darauf, 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 aufwendig eine solche Iteration geschätzt wird, was dann als Multiplikation aus Anzahl der Iterationen und Kosten pro Iteration die Gesamtkosten einer solchen Operation ergibt. Hierbei führt es häufig zu Irritationen und ist daher wichtig zu verstehen, dass insbesondere diese Mengen-Abschätzungen, 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 beim heutigen Stand des CBO 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 verweist der Autor auf seinen Artikel „Cost Based Optimizer Grundlagen", erschienen in der DOAG News, Ausgabe 06/2012, Seite 47, der auf dieses Thema ausführlicher eingeht.

Bevor es System-Statistiken gab, verwendete der CBO für die Kostenberechnung von Tabellen-Zugriffen (Full Table Scans) eine interne Formel, basierend auf dem eingestellten „DB_FILE_MULTIBLOCK_READ_COUNT"-Parameter, um die Kosten eines Full Table Scan im Vergleich zu einem Index-Zugriff zu berechnen. Da diese Berechnung auch kostenmäßig keine Unterscheidung zwischen Einzelblock- und Mehrfachblock-Zugriffen machte – es wurde einfach die Anzahl notwendiger Zugriffe gezählt, wobei ein Einzelblock-Zugriff in diesem Sinne äquivalent mit einem Mehrfachblock-Zugriff war und daher tendenziell Tabellen-Zugriffe gegenüber Index-Zugriffen 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 beziehungsweise reduzieren können. Dieser Ansatz hatte allerdings verschiedene Nachteile:

- Der eingestellte „DB_FILE_MULTIBLOCK_READ_COUNT" ist nur sehr indirekt eine Maßzahl für die tatsächliche Performance von Full Table Scan. In der Praxis kann zum einen die Anzahl der beim Full Table Scan tatsächlich pro I/O-Request gelesenen 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 mithilfe einer internen Formel entsprechend angepasst wurde – vereinfacht ausgedrückt, je größer der Wert für „DB_FILE_MULTIBLOCK_READ_COUNT" eingestellt ist, desto 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 intern stattdessen ein Wert von 16,41 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 *(siehe Abbildung 1)*.
- 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 Anwender 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.

- Die beiden Parameter wurden aufgrund des 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 – 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 allen diesen Erfahrungen und Gründen hat Oracle mit der Version 9i die System-Statistiken eingeführt, die diese Probleme lösen sollten. Insbesondere lassen sich darüber die Kosten für Full Table Scans nach oben korrigieren, 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, in denen zumindest der Einsatz von angepassten „OPTIMIZER_INDEX_CACHING"-Einstellungen aufgrund der Nicht-Berücksichtigung von Caching-Effekten bei der Kostenberechnung immer noch Sinn ergeben kann und das beschriebene Problem der Rundungen bei niedrigen Kosten für Index-Zugriffe umgangen wird – 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 Optimizer erzwungen wurde (per „OPTIMIZER_FEATURES_ENABLE"), änderten sich potenziell sehr viele Ausführungspläne und aufgrund der Vielzahl von Fehlerquellen, die die Berechnungen des CBO beeinflussen können, waren viele dieser Änderungen nicht positiv, es kam also bei vielen Kunden nach dem Upgrade zu Performance-Problemen. Diese Standard-System-Statistiken sind bis heute aktiv und können nur über undokumentierte Parameter oder ein 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 sie zu verwenden.

Wie System-Statistiken funktionieren

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

- Für Einzelblock- und Mehrfachblock-Zugriffe beschreiben die System-Statistiken, wie lange diese durchschnittlich benötigen („SREADTIM" und „MREADTIM" genannt), also eine Zeitangabe in Millisekunden. Dies löst als Seiteneffekt das beschriebene Problem, da jetzt die beiden Zugriffsarten im Sinne der berechneten Kosten unterschiedlich bewertet werden. Die Annahme ist, dass ein Mehrfachblock-Zugriff länger als ein Einzelblock-Zugriff 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 in Zeit auszudrücken – dies ist in der Spalte „TIME" eines Ausführungsplans bei Verwendung von System-Statistiken zu finden. Intern wird der Berechnungsmodus „CPU Costing" genannt, siehe nächster Punkt.
- Die CPU-Geschwindigkeit wird auch gemessen („CPUSPEED" beziehungsweise „CPUSPEEDNW") und eine CPU-Kosten- beziehungsweise Zeit-Komponente berechnet, basierend auf den durchgeführten Operationen. Jeder Zeilen- beziehungsweise Spalten-Zugriff benötigt CPU-Zeit, je nach Position der Spalte in einer Zeile auch unterschiedlich. Angewendete SQL-Funktionen (wie „UPPER", „LOWER", „TO_CHAR") 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 diejenigen 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 Mehrfachblock-Zugriff 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 stattdessen dieser gemessene Wert verwendet. Je nach Konfiguration (Parameter explizit gesetzt oder Standardwert) sowie 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 mehr dazu, wie diese Parameter die Kostenberechnung beeinflussen können.

Konfiguration der System-Statistiken

Wichtig: Nicht immer sind alle diese Messwerte verfügbar. Recht verwirrend

bei System-Statistiken ist die Tatsache, dass es unterschiedliche Arten gibt, je nachdem, auf welche Art und Weise sie ermittelt wurden („WORKLOAD“ und „NOWORKLOAD“). Für die Konfiguration von System-Statistiken steht ein umfangreiches Interface im „DBMS_STATS“-Paket zur Verfügung. Damit können sie 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 durch entsprechenden Aufruf im „DBMS_STATS“-Paket („GATHER_SYSTEM_STATS“) gemessen werden. Sie ermitteln nur einen eingeschränkten Umfang an Werten:

- CPU-Geschwindigkeit in Oracle-eigener Einheit („CPUSPEEDNW“)
- Durchschnittliche Zeit zur Festplatten-Kopfpositionierung in Millisekunden („IOSEEKTIM“)
- Durchschnittliche I/O-Datenübertragungsrate von Festplatte in Bytes pro Millisekunde („IOTFRSPEED“)
- Durchschnittliche Anzahl von Blöcken bei Mehrfachblock-Zugriff („MBRC“, nur im neuen „EXADATA“-Modus, später dazu mehr)

Die Messung von NOWORKLOAD-System-Statistiken erfolgt mit einer künstlichen Last, die die Datenbank bei der Messung selbst erzeugt – daher der Name „NOWORKLOAD“, da die Messung nicht auf einem Lastprofil basiert, die eine Applikation verursacht. 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. Dabei werden umfangreichere Messwerte ermittelt:

- CPU-Geschwindigkeit in Oracle-eigener Einheit („CPUSPEED“)
- Durchschnittliche Dauer des Einzelblock-Zugriffs in Millisekunden („SREADTIM“)
- Durchschnittliche Dauer des Mehrfachblock-Zugriffs in Millisekunden („MREADTIM“)
- Durchschnittliche Anzahl von Blöcken bei Mehrfachblock-Zugriff („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; erfolgten zum Beispiel keine Mehrfachblock-Zugriffe während einer entsprechenden Messung, sind sowohl der oben erwähnte „MBRC“ als auch „MREADTIM“ nicht in den Systemstatistiken verfügbar. Je nachdem, welche Messwerte fehlen oder auch bestimmten Überprüfungen nicht standhalten (wie zum Beispiel, dass „MREADTIM“ größer als „SREADTIM“ sein muss (siehe oben), was nicht immer in der Realität der Fall ist), werden diese entweder durch Standard-/Ersatz-Werte ersetzt oder die Art der Kostenberechnung fällt auf eine andere Art der Berechnung zurück („NOWORKLOAD“ anstatt „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 mithilfe einer indirekten Berechnung die Zeiten für Einzelblock- und Mehrfachblock-Zugriffe („SREADTIM“ und „MREADTIM“, berechnet durch Blockgröße, „IOSEEKTIM“ und „IOTFRSPEED“, siehe *Listing 1*).

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

Empfehlungen/Best Practices bezüglich System-Statistiken

Eines der Probleme im Umgang mit System-Statistiken ist, dass es recht unterschiedliche Aussagen gibt. Als sie

```
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
```

Listing 1: 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)

```
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 |
-------------------------------------------------------------------------------
```

Listing 2: Beispielhafte Kostenberechnung für einen Full Table Scan mit und ohne System-Statistiken

eingeführt wurden, gab es offizielle Empfehlungen seitens Oracle, sie 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 in seinem Buch „Troubleshooting Oracle Performance" System-Statistiken als Initialisierungsparameter ein und empfiehlt weiterhin grundsätzlich, diese als WORKLOAD-System-Statistiken zu ermitteln; er 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://sql-maria.com/2018/04/10/should-you-gather-system-statistics" und „https://blogs.oracle.com/optimizer/should-you-gather-system-statistics")*.

Eines der Probleme beim 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 es der Autor schon bei einigen Kunden vorgefunden hat, 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 potenzielle Instabilität in die Umgebung ein, auf die man möglichst verzichten sollte.

Eine Idee, die Christian Antognini auch als mögliche Alternative empfiehlt, ist die Variante, 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 jedoch von den meisten „DBMS_STATS"-Aufrufen unterstützt. Mit dieser Methode kann man die ermittelten Messwerte auch auf ihre Volatilität hin überprüfen und dann entscheiden, welche Werte am ehesten Sinn ergeben 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 CBO beeinflussen (System-Statistiken, Objekt-Statistiken, Art der Abfragen und Filter, eventuell Dynamic Sampling/Statistics etc.). Insofern stellen die System-Statistiken nur einen Teil davon dar. 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 zu weniger effizienten Ausführungsplänen kommt.

Der praktische Nutzen von System-Statistiken

Neben der Berücksichtigung der CPU-Komponente ist also die unterschiedliche und konfigurierbare Bewertung von Einzel- und Mehrfachblock-Zugriffen die wichtigste Neuerung, die mit System-Statistiken eingeführt wurde. Der Hintergedanke bei deren Einführung in Oracle 10g war offensichtlich, die Kostenberechnung des CBO Index-freundlicher zu gestalten. Dieses Ziel wurde grundsätzlich erreicht, so steigen die Kosten für einen Full Table Scan je nach eingestellter/gemessener durchschnittlicher Anzahl von Blöcken bei Mehrfachblock-Zugriff (MBRC) zwischen Faktor 1,8 und 7 im Vergleich zur Kostenberechnung ohne System-Statistiken *(siehe Listing 2)*; mehr Details dazu in der oben genannten Blog-Serie.

Wichtig ist in diesem Zusammenhang zu verstehen, dass es neben der angenommenen Anzahl von Blöcken pro Mehrfachblock-Zugriff (MBRC) maßgeblich das Verhältnis zwischen angenommener Zeit für Einzelblock- und Mehrfachblock-Zugriff („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 die durchschnittliche Anzahl der Blöcke bei Mehrfachblock-Zugriff (MBRC) niedriger eingestellt wird – auch ein manuelles Setzen von System-Statistiken per „DBMS_STATS.SET_SYSTEM_STATS" ist möglich.

```
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 |
-------------------------------------------------------------------------------------
```

Listing 3: Höhere Kosten bei der Berechnung von Parallel-Execution-Plänen und der Verwendung von „SLAVETHR"-Werten unterhalb des Standard-Werts

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:

- Wenn der CBO mithilfe des Parameters „OPTIMIZER_INDEX_COST_ADJ" Index-freundlicher konfiguriert werden soll, verringern sich die Kosten für Index-Zugriffe. Bei der Anpassung mit System-Statistiken erhöhen sich die Kosten für Full Table Scans – die Kosten für den Index-Zugriff bleiben unverändert.
- 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 beschriebenen Effekte einer möglichen Volatilität bei der Messung der System-Statistiken, des Zusammenspiels mit anderen Eingabewerten bei der Kostenberechnung (vor allem Mengen-Abschätzungen und Clustering-Faktor von Indizes) und auch der Tatsache, dass das Kostenmodell des CBO 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 Parallelverarbeitungs-Option der Enterprise Edition verwendet („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. Mit „SLAVETHR" lässt sich letzterer Faktor nach unten korrigieren, dem CBO also mitteilen, dass der Faktor niedriger als 0.9 liegt, die Parallelverarbeitung somit 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 potenziell favorisiert *(siehe Listing 3)*.

„MAXTHR" setzt der standardmäßigen, unendlichen Skalierbarkeit eine Obergrenze – ab einem bestimmten Grad an Parallelität verringern sich dann die berechneten Kosten nicht mehr, was durchaus sinnvoll ist und insbesondere zu verhindern hilft, dass unrealistisch hohe Parallelitätsgrade den CBO einen Ausführungsplan bevorzugen lassen, der in der Realität so nicht skalieren kann. Allerdings gilt dies nur für die Kostenberech-

```
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 |
-----------------------------------------------------------------------------------------
```

Listing 4: Berücksichtigung der limitierten Skalierbarkeit von Parallel Execution bei der Kostenberechnung durch „MAXTHR“

nung – 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 *(siehe Listing 4)*.,

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; bei der Berechnung der parallelen Kosten, die „SLAVETHR“ und „MAXTHR“ verwenden, wurde ebenfalls 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 Store, 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 Mehrfachblock-Zugriff) 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

```
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 |
---------------------------------------------------------------------------
```

Listing 5: Verringerte Kosten für einen Full Table Scan bei Verwendung von EXADATA-System-Statistiken

```
TIME represents cost multiplied by SREADTIM without I/O calibration, in this case here 2716 * 12 ms equals ap-
prox. 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 |
------------------------------------------------------------------------
```

Listing 6: Veränderte Zeit-(„TIME")Berechnung bei Vorhandensein von I/O-Kalibrierungswerten

viele Installationen, die den Parameter angeben, meistens aus historischen Gründen.

Ist 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 einen bestimmten Wert unterschreitet, also nicht sehr viele Prozesse für eine recht kleine SGA konfiguriert sind, dann wird bei der Ausführung versucht, typischerweise 1-MB-Mehrfachblock-Zugriffe 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 Full-Table-Scan-freundlicher macht. Damit sollten die Exadata und auch andere Systeme mit sehr schnellen Full Table Scans diese in den Ausführungsplänen eher favorisieren.

Sinn ergibt 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 *(siehe Listing 5)*.

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 Parallelisierungsgrads bei Parallel-Ausführungen. Diese Berechnung wiederum fußt maßgeblich auf dem durch I/O-Kalibrierung ermittelten I/O-Durchsatz. Auf den ersten Blick er-

```
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 de-
gree, 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
```

Listing 7: Verändertes Verhalten bei fehlenden I/O-Kalibrierungsergebnissen und Verwendung des „PARALLEL"-Hints auf Statement-Ebene ab Oracle 12c durch Verwendung/Annahme von Standardwerten für die I/O-Kalibrierung

scheint es allerdings merkwürdig, warum diese I/O-Kalibrierung zusätzlich zu den bereits bestehenden System-Statistiken eingeführt wurde – geht es doch bei beiden hauptsächlich 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 beziehungsweise deren Messung alle Ausführungspläne beeinflussen kann. Anstatt also das potenzielle Risiko einzugehen, Kunden, die das „Auto DOP"-Feature verwenden wollen, dazu zu zwingen, die System-Statistiken zu ermitteln und damit 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 CBO im Rahmen der Ausführungsplan-Erstellung, die vor der I/O-Kalibrierung eine direkte Relation zu den Kosten hatte (Kosten mal Zeit für einen Einzelblock-Zugriff („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 zehn Sekunden. Nur wenn also gemäß der neuen „TIME"-Berechnung eine Ausführung geschätzt länger als dieser Parameterwert dauert, wird Parallelverarbeitung in Betracht gezogen *(siehe Listing 6)*.

Wie die I/O-Kalibrierung funktioniert

Leider ist das Handling der I/O-Kalibrierung bei Weitem nicht so ausgereift wie bei den System-Statistiken, die über das „DBMS_STATS"-Paket ein umfassendes 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, sowie die maximale Latenz für einen Festplat-

tenzugriff. 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 daher zu einem Zeitpunkt durchgeführt werden, zu dem möglichst wenig Aktivität auf dem System ist. Dieser künstliche Benchmark erzeugt sowohl Einzelblock- als auch Mehrfachblock-Zugriffe und ermittelt darüber eine maximale „IOPS"-Rate, den maximalen I/O-Durchsatz und 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 „SLAVE-THR" 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, wie zum Beispiel Exadata, nicht wirklich repräsentative Werte ermitteln. Zusätzlich ist wichtig zu verstehen, dass Änderungen an der 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.

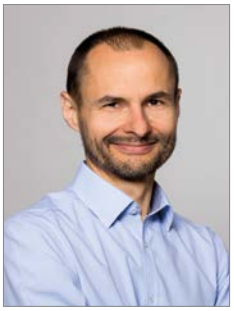
Ein weiterer signifikanter Unterschied ist das fehlende API zur Pflege der I/O-Kalibrierungswerte – so empfiehlt Oracle in Versionen vor 12c im Falle von ungünstig ermittelten Werten oder generell auf Exadata-Systemen beziehungsweise Systemen mit schnellem I/O, die I/O-Kalibrierungswerte manuell zu setzen. Allerdings ist dieses manuelle Setzen im Gegensatz zu den System-Statistiken, bei denen Werte direkt mit entsprechenden „DBMS_STATS.SET_SYSTEM_STATS"-Aufrufen 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 ein diesbezügliches offizielles API hilfreich wäre.

Neuerungen bei 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 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 durchgeführt. Damit man diesen Wert nicht manipulieren muss, wurde zusätzlich ein neuer Parameter „PARALLEL_DEGREE_LEVEL" eingeführt, mit dem man die „Auto DOP"-Berechnung entsprechend verändern kann. Der Standardwert beträgt „100", Werte kleiner als „100" ergeben niedrigere Parallelitätsgrade als normal, 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 kann *(siehe Listing 7)*.



Randolf Geist
randolf.geist@oracle-performance.de

Oracle erwirkt dauerhafte Verfügung gegen Rimini Street

Der Federal Court in Nevada hat entschieden, dass Oracle zu einer dauerhaften Verfügung gegen Rimini Street berechtigt ist. Der Drittanbieter für Software-Wartung und Support-Dienstleistungen wurde außerdem zur Zahlung von rund 30 Millionen US-Dollar verpflichtet.

Rimini Street habe seine Geschäfte aufgebaut, indem es die urheberrechtlich geschützte Software von Oracle verletzte, so das Gericht. Weiterhin erhielt Oracle in der letzten Phase des acht Jahre anhaltenden Rechtsstreits über 28 Millionen US-Dollar an Anwaltsgebühren zugesprochen. Die Fehde hatte ursprünglich damit begonnen, dass Oracle behauptete, Rimini Street sei durch illegalen Zugriff auf die technischen Support-Websites von Oracle an „massivem Diebstahl" beteiligt gewesen.

Weitere Informationen unter *„https://www.oracle.com/corporate/pressrelease/rimini-street-081418.html"*.