Workload-System-Statistiken sollen dem kostenbasierten Optimizer anzeigen, was er von der Hardware erwarten darf. Der Administrator muss diese in Eigeninitiative ermitteln. Doch wie geht man dabei am besten vor?

Glücksspiel „System-Statistiken“ – das Märchen vom typischen Workload

Thorsten W. Grebe

Damit der kostenbasierte Optimizer bestmögliche Ausführungspläne berechnen kann, benötigt er diverse Statistiken. Am wichtigsten sind die Objekt-Statistiken, mit denen die aktuelle Befüllung von Tabellen und Indices charakterisiert wird und die im günstigen Normalfall automatisch durch Wartungsjobs der Datenbank aktualisiert werden. Diese Objekt-Statistiken sind jedoch blind gegenüber der eingesetzten Hardware – sie behandeln alle CPUs und Festplatten gleich. Um dem Administrator eine Möglichkeit zu geben, trotzdem die Gewichtung von Index-Zugriffen zu Indexrange- oder Table-Scans je nach Leistungsfähigkeit der Hardware besser austarieren zu können, führte Oracle zusätzliche Init-Parameter ein, von denen die bekanntesten „optimizer_index_cost_adj“ und „optimizer_index_caching“ sind. Diese einfache, aber effiziente Methode der Optimizer-Beeinflussung wurde ab der Version 9*i* mit Einführung des kostenbasierten Optimizers durch die sogenannten „Workload-Statistiken“ abgelöst, die genau die fehlende Wissenslücke des Optimizers über die zugrunde liegende Hardware füllen sollte. Dass sich dieses modernere, im Grunde überlegene Verfahren bis heute nicht flächendeckend verbreitet hat, dürfte daran liegen, dass der DBA von sich aus aktiv werden muss. Erschwerend für die Akzeptanz kommt die Aura hinzu, die System-Statistiken umgibt, und die Mühe, die sie verursachen, um nachvollziehbare System-Statistiken zu ermitteln und die System-Konfiguration nicht durch einen Schnellschuss zu verschlechtern.

System-Statistiken sollen dem Optimizer Kenndaten für die Hardware liefern, auf der die Datenbank-Instanz läuft: Wie schnell die CPU, wie träge der Zugriff auf den Datenträger oder wie hoch der zu erwartende I/O-Durchsatz ist. In einer neu installierten Instanz sind diese Werte unbekannt. Die Datenbank setzt Vorgabewerte ein, die sogenannten „Default-NoWorkload-Statistiken“. Nachgeschlagen werden können sie in der Tabelle „sys.aux_stats$“. Abbildung 1 zeigt diese Tabelle einer frischen Instanz mit ergänzten Zeilennummern und Kommentaren. Nur drei Statistik-Werte sind hier gesetzt (siehe Spalte „PVAL1“, Zeile 5-7): „CPUSPEEDNW“, „IOSEEKTIM“ und „IOTFRSPEED“ – wie schnell die CPU, wie latent ein Einzelblockzugriff und wie hoch der zu erwartende IO-Durchsatz ist. Die meisten Systeme arbeiten sehr gut mit den Vorgaben. Von den drei Werten wird lediglich „CPUSPEEDNW“ zum Zeitpunkt des Eintrags in „aux_stats$“ bestimmt und „IOSEEKTIM“ und „IOTFRSPEED“ werden auf 10 ms beziehungsweise 4 MB/s gesetzt.

```
SQL> select rownum as rn, a.* from aux_stats$ a;

RN SNAME          PNAME      PVAL1 PVAL2          Kommentar
-- -------------- ---------- ----- -------------- --------------------------------
 1 SYSSTATS_INFO  STATUS           COMPLETED      oder: MANUAL-, AUTOGATHERING
 2 SYSSTATS_INFO  DSTART           08-29-2012 18:47 Messung-Startzeit
 3 SYSSTATS_INFO  DSTOP            08-29-2012 18:47 Messung-Ende,bei NoWorkload-DSTART
 4 SYSSTATS_INFO  FLAGS         1                0=gelöscht, 1=gesetzt
 5 SYSSTATS_MAIN  CPUSPEEDNW 1306                NoWorkload: Mio Ops/Sek pro CPU
 6 SYSSTATS_MAIN  IOSEEKTIM    10                NoWorkload: Zugriffszeit in ms
 7 SYSSTATS_MAIN  IOTFRSPEED 4096                NoWorkload: Durchsatz in KB/sec
 8 SYSSTATS_MAIN  SREADTIM                       Zugriffszeit in ms für Einzelblock
 9 SYSSTATS_MAIN  MREADTIM                       Zugriffszeit in ms für Multiblock
10 SYSSTATS_MAIN  CPUSPEED                       Workload: Mio Ops/Sek pro CPU
11 SYSSTATS_MAIN  MBRC                           Durschn. Zahl Blöcke bei FTS, FFS
12 SYSSTATS_MAIN  MAXTHR                         Max. Dursatz in Byte/sec (PX-Proz.)
13 SYSSTATS_MAIN  SLAVETHR                       Durschn. Dursatz in B/s (PX-Proz.)
```

Abbildung 1: Die Tabelle „aux_stats$“. Auch in aktuellen Installationen (11.2.0.3) dürfte man hier meist „Werks-Einstellungen“ vorfinden. Eine Zugriffszeit von 10 ms und ein Durchsatz von 4MB/s waren vor zehn Jahren passable Annahmen.

NoWorkload-System-Statistiken

Neben diesen Default-NoWorkload-System-Statistiken, über die jede Datenbank ab der Version 10*g* verfügt, unterscheidet Oracle noch die sogenannten „NoWorkload-Statistiken“ sowie die Workload-Statistiken. „NoWorkload-Statistik“ bedeutet, dass der Administrator einen künstlichen Benchmark erzeugt, über den reale Werte für „IOSEEKTIM“ und „IOTFRSPEED“ ermittelt werden sollen. Der Benchmark ist sehr einfach über das Kommando „exec dbms_stats.gather_system_stats()“ (ohne Argumente) auszuführen. Je nach Anzahl vorhandener Daten-Dateien läuft die Prozedur einige Sekunden bis einige Minuten. Gemessen werden CPU-Leistung und Zugriffszeiten für lesende Zugriffe auf alle Daten-Dateien. Blickt man nach dem

Ablauf der Prozedur erneut in die Tabelle „aux_stats$“, sind „DSTART“ und „DSTOP“ auf die Startzeit des Laufs gesetzt und „CPUSPEEDNW“, „IOSEEKTIM“ und „IOTFRSPEED“ neu ermittelt. Der Optimizer verwendet ab sofort für die Berechnung von Ausführungsplänen diese neuen Werte. Stellt man fest, dass die Werte unrealistisch sind, lassen sie sich über das Kommando „exec dbms_stats.delete_system_stats“ wieder entfernen: Dabei werden „IOSEEKTIM“ und „IOTFRSPEED“ zurück auf „10“ beziehungsweise „4096“ gesetzt, „CPUSPEEDNW“ wird in der Regel neu ermittelt. Bei moderner Hardware sollte „IOSEEKTIM“ deutlich unter 10 ms liegen, „IOTFRSPEED“ deutlich über dem sehr konservativen Default von 4 MB/s, etwa bei 20-40. „CPUSPEEDNW“ bezieht sich auf die Leistung einer einzelnen CPU. Oracle zählt hierbei interne Operationen, die nicht dokumentiert sind. Der erhaltene Wert sollte in der Nähe dessen liegen, was bei Linux etwa das Kommando „cat /proc/cpuinfo“ oder bei Windows das Kommando „wmic cpu get CurrentClockSpeed“ zurückliefert.

Bereits bei der recht einfachen und für den DBA komfortablen Ermittlung von NoWorkload-Statistiken werden einige Schwächen des beschriebenen Vorgehens offenbar. Die ermittelten Werte können sehr stark schwanken. Ungültige Messungen von NoWorkload-Statistiken sind nicht durch den Eintrag „BADSTATS“ in der Spalte STATUS erkenntlich, sondern daran, dass für „IOTFRSPEED“ nach einer Messung wieder der Defaultwert „4096“ eingetragen ist. Dies geschieht beispielsweise dann, wenn die Festplatten zeitgleich zur NoWorkload-Messung Lese- oder Schreib-Vorgänge ausführen – häufig liegt dann auch „CPUSPEEDNW“ deutlich unter dem Erwarteten.

Ob bei einer einzelnen Messung auf Anhieb vernünftige Werte ermittelt werden, ist Glückssache. Und ob er Glück hatte, kann nur derjenige erfahren, der Messreihen erstellt und die erhaltene Streuung der Werte betrachtet. Wie dies aussehen kann, ist im Folgenden anhand der Workload-Statistiken demonstriert.

Workload-System-Statistiken

Die dritte Spielart der System-Statistiken sind die sogenannten „Workload-System-Statistiken“. Im Gegensatz zu den NoWorkload-System-Statistiken wird kein künstlicher Benchmark erzeugt, sondern Oracle versucht, reale Werte aus dynamischen Performance-Views zu ermitteln. Ob in „aux_stats$“ Workload- oder NoWorkload-Statistiken eingetragen sind, erkennt man daran, dass alle oder einige der Werte in den Zeilen 8-13 (siehe Abbildung 1) gesetzt sind. Workload-Statistiken sind aus Oracle-Sicht den NoWorkload-Statistiken vorzuziehen, weil sie detaillierter Auskunft geben: Es wird zwischen Einzelblock- und Mehrblock-Zugriffen unterschieden („SREADTIM“ vs. „MREADTIM“). Ferner wird ermittelt, wie viele Blöcke bei einem Segment-Scan tatsächlich in der Regel zusammenhängend gelesen werden können („MBRC“) und wie viele CPU-Zyklen tatsächlich geliefert worden sind („CPUSPEED“). Außerdem wird der Durchsatz von parallelen Abfragen gemessen. Die sechs Werte der Workload-Statistiken sind in Abbildung 2 zusammengefasst.

Für die Berücksichtigung dieser Workload-Statistikwerte durch den Optimizer gilt eine einfache Regel: Wenn Workload-Statistiken existieren, verwendet der Optimizer diese und ignoriert NoWorkload-Statistiken. Dabei gilt die etwas schwierigere Zusatzregel: Wenn Workload-Statistiken oder Teile davon als ungültig diagnostiziert werden, dann verfällt der Optimizer in diverse, undokumentierte Sonder-Routinen. Ein bekannter Fall ist der unten beschriebene, bei dem für Multiblock-Zugriffe kürzere Zeiten gemessen werden als für Einzelblock-Zugriffe.

Wie kommt man an ordentliche Workload-System-Statistiken? Laut dem „11*g* Upgrade Best Practices Guide“ ist dies so einfach, wie in Abbildung 3 dargestellt: Messung starten – Schuss – Messung – fertig. Ähnlich kurze Anweisungen finden sich im „Upgrade Companion“ und „Performance Tuning Guide“.

Lucky Luke und John Wayne kommen einem dabei in den Sinn. Beide beherrschten den Schuss aus der Hüfte und trafen mit dieser Technik zielsicher fliegende Zehn-Cent-Stücke auf hundert Yard Entfernung. Nur sehr wenige wissen, dass der Hüftschuss schon im Wilden Westen nicht funktionierte, was erklärt, warum kaum jemand stutzig wird, wenn Oracle hier eine vergleichbare Technik vorschlägt.

`SREADTIM`	Zugriffsgeschwindigkeit in Millisekunden für den Einzelblockzugriff, z.B. bei Indexzugriffen.
`MREADTIM`	Zugriffsgeschwindigkeit in Millisekunden für den Multiblockzugriff, z.B. bei *Full Table Scans* (FTS) oder *Fast Full Index Scans* (FFS); MREADTIM <= 1,2 * SREADTIM sind invalide und lassen den Optimizer Zugriffskosten im Version 8i-Modus kalkulieren (→ MOS 153761.1).
`CPUSPEED`	Millionen Oracle-Operationen pro Sekunde, die eine einzelne CPU verrichtet.
`MBRC`	Wie viele Blöcke werden bei einem Multiblock-Zugriff tatsächlich gelesen; finden während des Messzeitraums keine Segmentscans statt, bleibt der Wert leer; dann wird stattdessen MBRC=8 gesetzt, abgeleitet von `_db_file_optimizer_read_count`; dieser Parameter steht per Default auf 8 bis zu dem Tag, an dem der DBA `db_file_multiblock_read_count` anfasst.
`MAXTHR`	Wie hoch ist der maximal gemessene Durchsatz der Storage in Bytes/Sekunde, der bei parallelen Abfragen erreicht wird; (andere Einheit als `IOTFRSPEED`); ohne parallele Prozesse bleibt der Wert leer. Bedeutung für CBO unklar.
`SLAVETHR`	Wie hoch ist der gemessene Datendurchsatz von einzelnen Slave-Prozessen bei parallelen Abfragen in Bytes/Sekunde – werden während des Messintervalls keine parallelen Abfragen ausgeführt, bleibt der Wert leer (meist ca. 2 Größenordnungen kleiner als `MAXTHR`). Bedeutung für CBO unklar.

Abbildung 2: Die sechs Werte der Workload-Statistiken. Die ersten vier dieser Werte sollten sinnvoll gesetzt werden. Welche Rolle „MAXTHR“ und „SLAVETHR“ in 11.2.0.3 bei der Kostenberechnung spielen, ist nicht offensichtlich.

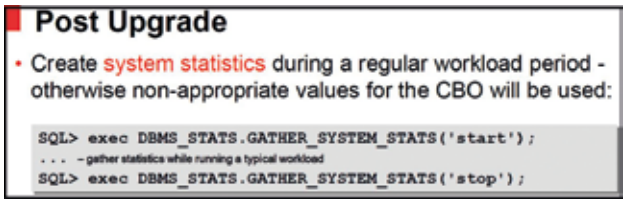


Abbildung 3: Kurz und bündig – die Kurzanleitung von Oracle. Wer so vorgeht, braucht viel Glück bei der Messung.

```
SQL> begin
       dbms_stats.create_stat_table(
          ownname => 'SYSTEM',
          stattab => 'SYSTEM_STATS_TWG'
          );
       end;
       /
```

Abbildung 4: Erstellen der Sammeltabelle. Messungen werden erst gesammelt und analysiert, bevor man sie in „aux_stats$“ einträgt. Schema und Tabellen-Name sind frei wählbar.

```
begin
 dbms_scheduler.create_job(
  job_name     => 'twg_sys_stats',
  job_type     => 'plsql_block',
  job_action   => 'declare
                   l_start varchar2(14);
                   l_label varchar2(30);
                   begin
                    select to_char(sysdate, ''yyyymmddhh24mi'')
                      into l_start from dual;
                   l_label := ''Workload ''
                           || SYS_CONTEXT(''userenv'', ''instance_name'')
                           || ''_''
                           || l_start;
                   dbms_stats.gather_system_stats(
                               gathering_mode => ''INTERVAL'',
                               interval       => 10,
                               statid         => l_label,
                               stattab        => ''SYSTEM_STATS_TWG'',
                               statown        => ''SYSTEM'');
                   end;',
  start_date   =>  sysdate,
  end_date     =>  sysdate+35,
  repeat_interval => 'FREQ=MINUTELY;INTERVAL=11' ,
  enabled      =>  true,
  comments     =>  'Workload Systemstatistiken, DOAG News 6-2012');
END;
/

-- für NoWorkload-Statistiken
dbms_stats.gather_system_stats(
 gathering_mode => ''NOWORKLOAD'',
 statid     => l_label,
 stattab    => ''SYSTEM_STATS_TWG'',
 statown    => ''SYSTEM'');
-- evtl anpassen, z.B.:
' ' NoWorkload_' '
-- für NoWorkload-Statistiken verkürzen, z.B.:
sysdate + 1/12
```

Abbildung 5: Statistik-Job aufsetzen. Der PL/SQL-Block in der Job-Action sammelt Workload-Statistiken für 35 Tage. Alle 11 Minuten wird ein 10-minütiges Mess-Intervall gestartet. Durch Austausch des Codes in den Boxen lassen sich NoWorkload-Statistiken sammeln. Für NoWorkload-Statistiken sollte die Dauer begrenzt werden, da diese das System durch einen künstlichen Workload belasten (hier „sysdate + 1/12“ = 2 Stunden).

Auch der Hinweis auf den „typical workload“ suggeriert eine nicht existierende Genauigkeit. Abgesehen davon, dass der Begriff abstrakt und ein Workload typisch für den Morgen, den Tag, die Woche oder den Monat ist, mag man spitzfindig anmerken, dass ein Workload stets typisch für seinen Zeitraum ist – dies liegt im Wesen des „typischen Workload“.

Das Kernproblem beim Erfassen von Workload-Statistiken liegt in dem Messverfahren, das Oracle verwendet. Einerseits ist die Idee hinter der verwendeten Methode die beste, die man sich ausdenken kann. Oracle misst während der Aufzeichnung das, was tatsächlich stattfindet. Workload-Statistiken sollen dem Optimizer nicht sagen, wie hoch die zugelassene Höchstgeschwindigkeit der Hardware ist, sondern was tatsächlich unter Berücksichtigung aller Widrigkeiten bei der Instanz an Ressourcen ankommt.

Eine Schwierigkeit bei diesem Ansatz ist jedoch, dass während einer Flaute keine Messdaten entstehen. Wenn keine Segment-Scans stattfinden (oder nur „direct reads“), lassen sich keine Werte für „MBRC“ und „MREADTIM“ ermitteln. Finden keine parallelen Ausführungen statt, können keine Werte für „MAXTHR“ und „SLAVETHR“ gemessen werden. Ist das Mess-Intervall zu lang, erhält man nichtssagende, niedrige Mittelwerte. Wählt man zur falschen Zeit ein zu kurzes Intervall, dann misst man gar nichts oder irreführende Ausreißer, etwa I/O-Stürme oder kurzfristige Überlastungen. Das Messen gleicht dem Schießen aus der Hüfte. Ein weiteres, ernsteres Problem des skizzierten Verfahrens liegt darin, dass das Ergebnis der Zufalls-Messung direkt in die Tabelle „aux_stats$“ eingetragen und damit sofort für alle Kostenberechnungen des Optimizers aktiv wird.

Wer zuverlässige Workload-Statistiken ermitteln will, dem bleibt nichts anderes übrig, als sich Zeit zu nehmen und selbst Mess-Reihen aufzuzeichnen. So einfach, wie es die Anleitung in Abbildung 3 suggeriert, geht es nicht. Das Ermitteln von Workload-Statistiken sollte in sechs Schritten erfolgen:

1. Statistik-Sammeltabelle erstellen
2. Scheduler-Job einrichten, der Zehn-Minuten-Intervalle aufzeichnet
3. Daten grafisch aufbereiten
4. Mit gesundem Menschenverstand die sechs Workload-Statistikwerte festlegen
5. Plausibilitätsregeln beachten
6. Workload-Statistiken manuell in die Tabelle „aux_stats$“ eintragen

Statistik-Sammeltabelle erstellen

Es sollen mehrere Messungen von Workload-Statistiken vorgenommen, die Ergebnisse sollen jedoch erst gesammelt und analysiert werden, bevor sie für den Optimizer sichtbar in der Tabelle „aux_stats$“ erscheinen. Für solche Zwecke sieht Oracle eine generische Sammel-Tabelle vor, die über „dbms_stats.create_stat_table“ angelegt wird (siehe Abbildung 4).

Scheduler-Job einrichten

Wie lang sollte eine Messung dauern? Für eine Mess-Reihe gilt generell: Je kürzer die Mess-Intervalle, desto besser. Fünf- bis zehnminütige Intervalle liefern gute Ergebnisse. Bei Intervallen über zwanzig Minuten werden Konturen verwischt und Spitzenwerte weggemittelt. Wer mehrstündige Intervalle misst, kann gleich zum Würfel greifen. Das Listing in Abbildung 5 legt einen Scheduler-Job an, der einen Monat lang zehnminütige Mess-Intervalle durchführt. Die Belastung des Systems während dieses Zeitraums durch die Messung ist vernachlässigbar. Mit den Statements in Abbildung 6 wird geprüft, ob der Job erfolgreich eingerichtet wurde und ob Statistiken gesammelt werden.

Daten grafisch aufbereiten

Nachdem ausreichend Datensätze gesammelt worden sind, ist die Zeit reif für die Auswertung des Zahlen-Materials. Hier liegt der nächste Stolperstein, denn die Feldbelegungen von „aux_stats$“ und der Sammel-Tabelle (hier: „system_stats_twg“) passen nicht zueinander. Die Korrelation der Feldbelegung ist weder dokumentiert noch offensichtlich. Der von Oracle dokumentierte Weg für den Übertrag der Werte aus der Sammeltabelle nach

```
SQL> select job_name
          , run_count as cnt
          , failure_count as fail
          , last_start_date
       from dba_scheduler_jobs
      where job_name = 'TWG_SYS_STATS'
         or job_name like 'ST$%';

JOB_NAME      CNT FAIL LAST_START_DATE
------------ --- ---- ---------------------------------
TWG_SYS_STATS   1    0 30.09.12 11:44:43,012000 +02:00
ST$637          0    0

SQL> select statid
          , c1 as status
          , c2 as startzeit
          , c4 as messart
       from system.SYSTEM_STATS_TWG
      order by statid;

STATID                        STATUS       STARTZEIT        MESSART
----------------------------- ------------ ---------------- ---------
WORKLOAD_ORCL_201209301144 COMPLETED     09-30-2012 11:44 CPU_SERIO
WORKLOAD_ORCL_201209301144                                  PARIO
WORKLOAD_ORCL_201209301155 AUTOGATHERING 09-30-2012 11:55 CPU_SERIO
WORKLOAD_ORCL_201209301155                                  PARIO
```

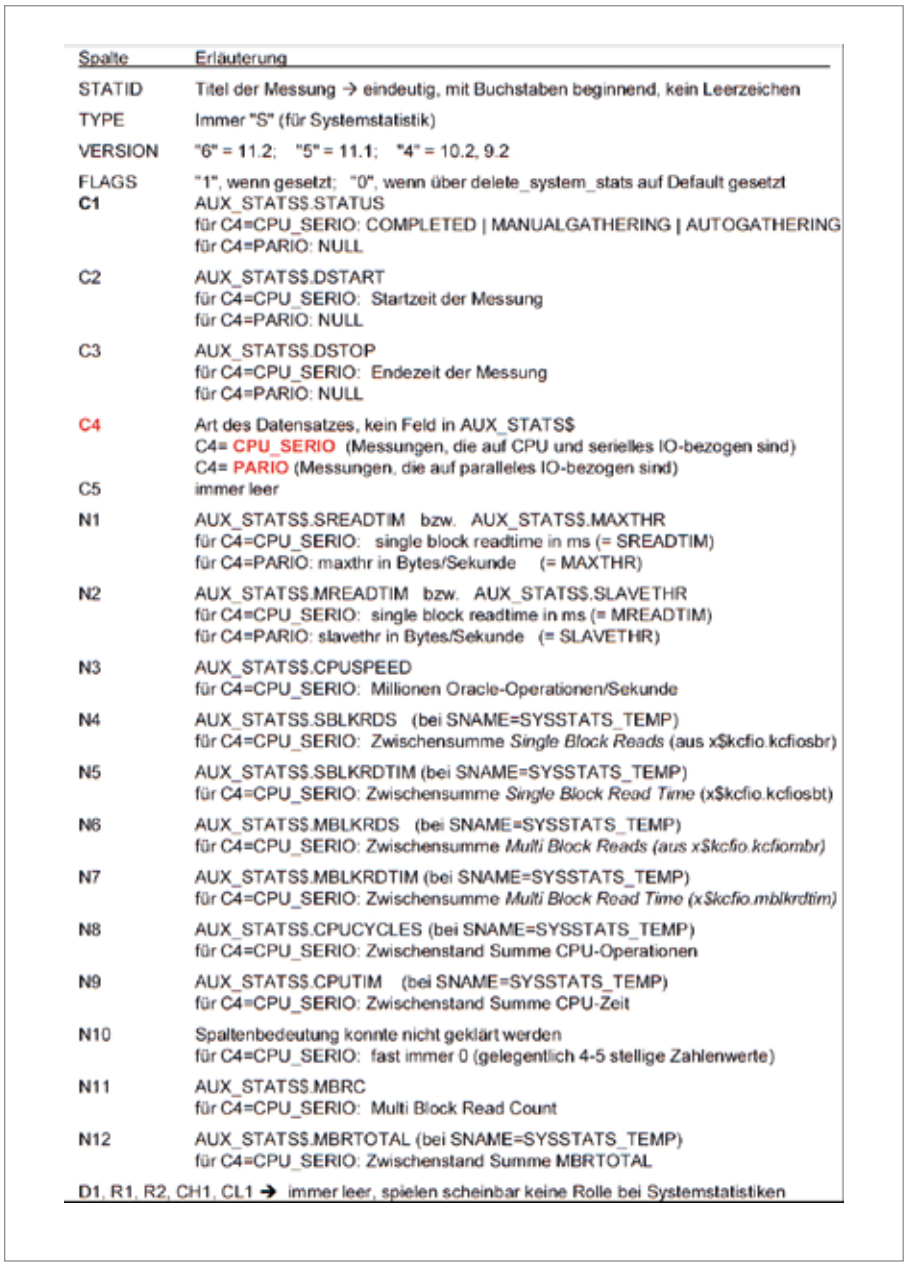
Abbildung 6: Prüfen, ob der Statistik-Job erfolgreich angelegt wurde und Statistiken gesammelt werden. Während eines Mess-Intervalls erscheint ein interner Job mit einer fortlaufenden Nummer, hier „ST$637“. Pro Messung kommen zwei Datensätze in die Sammel-Tabelle.

„sys.aux_stats$“ erfolgt über die Prozedur „dbms_stats.set_system_stats“. Das ist in Ordnung für das Setzen von einzelnen Werten, jedoch für unsere Zwecke ungeeignet, denn die Qualität mehrerer Tausend Datenpunkte soll evaluiert werden. Das Mapping ist ein mühseliger, glücklicherweise aber nur einmaliger Vorgang (siehe Abbildung 7). Oracle scheint hier seit einigen Versionen am Mapping keine Veränderungen vorgenommen zu haben (für NoWorkload-Statistiken ist die Korrelation in Abbildung 8 erläutert). Mit den beiden SQL-Statements in Abbildung 9 lassen sich über den SQL-Developer die Messdaten direkt in ein Excel-taugliches Format exportieren und anschließend in einem Tabellenkalkulationsprogramm öffnen. Abbildung 10 (siehe Seite 57) zeigt auf diese Weise erstellte Graphen beispielhaft für „SREADTIM“ und „MREADTIM“.

Mit gesundem Menschenverstand Workload-Statistikwerte festlegen

Die beste Methode, die Qualität der Daten zu überprüfen, ist die optische. Man betrachtet den Kurvenverlauf der Graphen und zwar für jeden Wert einzeln, also sechs Graphen. Das macht bei zehn Datenbanken bereits sechzig Graphen und stellt spätestens hier den Ansatz in Frage. Wer soll das leisten? Lohnt sich dieser Aufwand? Warum greift Oracle hier nicht mit einem Advisor unter die Arme? Allein die beiden Graphen für „MREADTIM“ und „SREADTIM“ (siehe Abbildung 10) werfen Fragen auf, statt gesicherte Antworten zu geben.

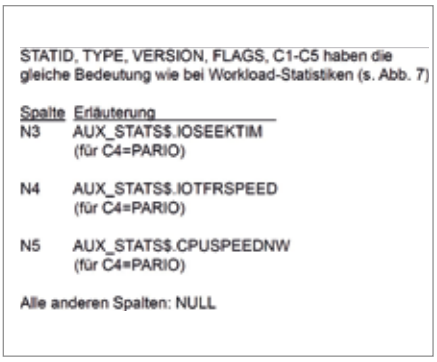
Zunächst fällt auf, dass es tatsächlich so etwas wie einen typischen Workload gibt, der sich täglich wiederholt. Aber wann hätte man hier messen sollen? Es gibt regelmäßige Ballungen von „MREADTIM“, in denen „SREADTIM“ entweder rasend schnell ist oder kaum vorhanden und daher weggemittelt. Die beiden Graphen legen nahe, dass beide Werte getrennt berechnet werden sollten – wobei man gleich in das nächste Dilemma läuft. Die Werte, die für „SREADTIM“ gemessen werden, sind kontinuierlich höher als diejenigen für „MREADTIM“. Die Sinnfrage



Spalte	Erläuterung
STATID	Titel der Messung → eindeutig, mit Buchstaben beginnend, kein Leerzeichen
TYPE	Immer "S" (für Systemstatistik)
VERSION	"6" = 11.2; "5" = 11.1; "4" = 10.2, 9.2
FLAGS	"1", wenn gesetzt; "0", wenn über delete_system_stats auf Default gesetzt
C1	AUX_STATS$.STATUS für C4=CPU_SERIO: COMPLETED | MANUALGATHERING | AUTOGATHERING für C4=PARIO: NULL
C2	AUX_STATS$.DSTART für C4=CPU_SERIO: Startzeit der Messung für C4=PARIO: NULL
C3	AUX_STATS$.DSTOP für C4=CPU_SERIO: Endezeit der Messung für C4=PARIO: NULL
C4	Art des Datensatzes, kein Feld in AUX_STATS$ C4= CPU_SERIO (Messungen, die auf CPU und serielles IO-bezogen sind) C4= PARIO (Messungen, die auf paralleles IO-bezogen sind)
C5	immer leer
N1	AUX_STATS$.SREADTIM bzw. AUX_STATS$.MAXTHR für C4=CPU_SERIO: single block readtime in ms (= SREADTIM) für C4=PARIO: maxthr in Bytes/Sekunde (= MAXTHR)
N2	AUX_STATS$.MREADTIM bzw. AUX_STATS$.SLAVETHR für C4=CPU_SERIO: single block readtime in ms (= MREADTIM) für C4=PARIO: slavethr in Bytes/Sekunde (= SLAVETHR)
N3	AUX_STATS$.CPUSPEED für C4=CPU_SERIO: Millionen Oracle-Operationen/Sekunde
N4	AUX_STATS$.SBLKRDS (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischensumme *Single Block Reads* (aus x$kcfio.kcfiosbr)
N5	AUX_STATS$.SBLKRDTIM (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischensumme *Single Block Read Time* (x$kcfio.kcfiosbt)
N6	AUX_STATS$.MBLKRDS (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischensumme *Multi Block Reads (aus x$kcfio.kcfiombr)*
N7	AUX_STATS$.MBLKRDTIM (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischensumme *Multi Block Read Time (x$kcfio.mblkrdtim)*
N8	AUX_STATS$.CPUCYCLES (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischenstand Summe CPU-Operationen
N9	AUX_STATS$.CPUTIM (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischenstand Summe CPU-Zeit
N10	Spaltenbedeutung konnte nicht geklärt werden für C4=CPU_SERIO: fast immer 0 (gelegentlich 4-5 stellige Zahlenwerte)
N11	AUX_STATS$.MBRC für C4=CPU_SERIO: Multi Block Read Count
N12	AUX_STATS$.MBRTOTAL (bei SNAME=SYSSTATS_TEMP) für C4=CPU_SERIO: Zwischenstand Summe MBRTOTAL

D1, R1, R2, CH1, CL1 → immer leer, spielen scheinbar keine Rolle bei Systemstatistiken

Abbildung 7: Korrelation zwischen der Statistik-Tabelle und „aux_stats$" für Workload-Statistiken. Pro Messung entstehen zwei Datensätze, einer für Paralleles (C4=PARIO), einer für alles andere (C4=CPU_SERIO). Die Felder „N4" bis „N9" und „N12" speichern Zwischenstände. Diese Werte sind in „aux_stats$" nur sichtbar, wenn keine Statistik-Tabelle verwendet wird. Für NoWorkload-Statistiken gilt eine andere Belegung (siehe Abbildung 8).



STATID, TYPE, VERSION, FLAGS, C1-C5 haben die gleiche Bedeutung wie bei Workload-Statistiken (s. Abb. 7)

Spalte	Erläuterung
N3	AUX_STATS$.IOSEEKTIM (für C4=PARIO)
N4	AUX_STATS$.IOTFRSPEED (für C4=PARIO)
N5	AUX_STATS$.CPUSPEEDNW (für C4=PARIO)

Alle anderen Spalten: NULL

Abbildung 8: Korrelation zwischen der Statistik-Tabelle und „aux_stats$" für NoWorkload-Statistiken. Die drei Werte der NoWorkload-Statistiken stecken in den Spalten „N3", „N4" und „N5" für die Datensätze „C4=PARIO". Die Unterscheidung in „PARIO" für paralleles I/O und „CPU_SERIO" für Messwerte, die sich auf CPU und serielles I/O beziehen, wurde für Workload-Statistiken in 9i eingeführt. Für die erst in 10g hinzugekommenen NoWorkload-Statistiken wurde diese Unterscheidung ignoriert, die drei NoWorkload-Statistikwerte landen in den „PARIO"-Datensätzen.

```
SQL> select statid
          , n1 as maxthr
          , n2 as slavethr
       from system_stats_twg
      where statid like '%_ORCL%'
        and c4 = 'PARIO'
      order by STATID ;

SQL> select statid
          , c2 as Beginn
          , c3 as Ende
          , n1 as sreadtim
          , n2 as mreadtim
          , n3 as cpuspeed
          , n11 as mbrc
       from system_stats_twg
      where statid
       like '%_ORCL%'
        and C4 = 'CPU_SERIO'
        and C1 = 'COMPLETED'
      order by STATID ;
```

Abbildung 9: Auslesen der Statistikwerte aus der Sammel-Tabelle für Workload-Statistiken. Das Ergebnis der beiden Abfragen kann der SQL Developer direkt in ein Excel-taugliches Format exportieren.

gärt. Denn wenn „MREADTIM" ähnlich groß ist wie „SREADTIM", dann verfällt der Optimizer in den archaischen Modus und rechnet Block-Zugriffe im Stil von Oracle 8*i* (MOS 153761.1). Die genaue Grenze des Umschlags liegt je nach System zwischen dem 1,0- und 1,2-fachen von „SREADTIM". Was unlogisch klingt – dass „SREADTIM" größer sein sollte als „MREADTIM" –, ist heute häufig die beobachtete Regel. Möglich machen dies „Read-ahead"-Algorithmen moderner SANs.

Im Zusammenhang mit Workload-Statistiken liest man gelegentlich den wohlgemeinten Appell: „Tell the optimizer the truth!". Gerne wollten wir dem Optimizer die Wahrheit sagen, allein er verträgt sie nicht. Der Zeitpunkt

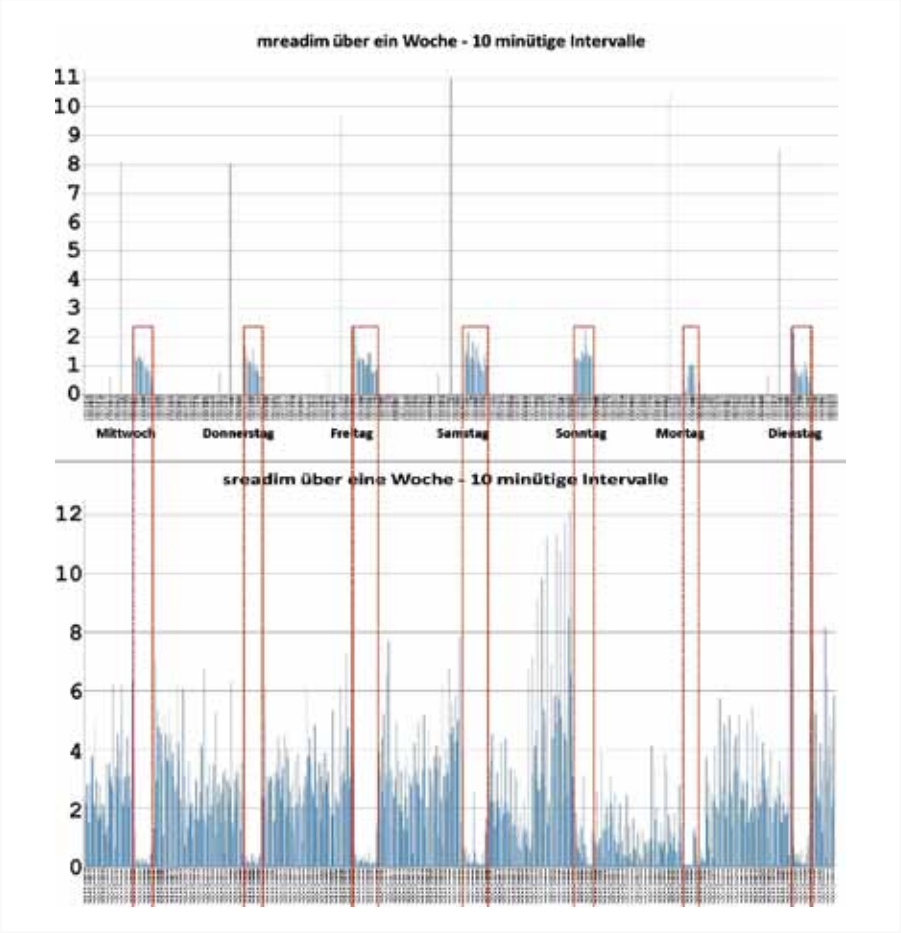


Abbildung 10: „SREADTIM vs. MREADTIM" in 10-minütigen Intervallen über eine Woche gemessen (in ms). Nachsinnen über das Wesen des „typischen Workload": Wo „MREADTIM" ist, ist „SREADTIM" nicht – zumindest in dieser Datenbank. Sind „single reads" tatsächlich so schnell in den Tälern oder sind es bloß so wenige, dass die Mittelwerte täuschend niedrig ausfallen? In welchem Bereich der Grafik wäre man bei einer einzelnen Zufallsmessung gelandet?

für Pragmatismus ist gekommen: Wir bestimmen zuerst einen Wert für „SREADTIM", dann schummeln wir „MREADTIM" so hin, dass wir nicht in die Invaliditäts-Falle tappen.

Das Ermitteln von „SREADTIM" kann entweder optisch durch Abschätzen aus dem Graphen (siehe Abbildung 10) erfolgen oder mathematisch aus den gemessenen Werten berechnet werden.

Abbildung 11 zeigt eine mathematisch Variante, die brauchbare Ergebnisse liefert. Sie errechnet für „SREADTIM" einen angepassten Mittelwert von 2,25 ± 0,04. Um unsere Statistikwerte nicht durch ein zu niedriges „MREADTIM" zu entwerten, setzen wir es auf das 1,3-fache von „SREADTIM", also: „MREADTIM = 1,3*SREADTIM = 2,25 * 1,3 = 2,9". Wir verlassen das Gebiet der reinen Wissenschaft.

Auch für die Bestimmung von „MBRC" und „CPUSPEED" sollten die entsprechenden Graphen inspiziert werden. Man kann nicht stur auf die Mathematik vertrauen, ohne den errechneten Wert mit dem Graphen-Verlauf verglichen zu haben. Kurios sind „MAXTHR" und „SLAVETHR" – es ist unklar, wie und ob diese überhaupt in die Berechnungen des Optimizers eingehen. Starke Wertschwankungen bei „MAXTHR" und „SLAVETHR" sind nicht ungewöhnlich und lassen zweifeln, welcher Wert hier sinnvoll einzutragen wäre – weder Mittelwert noch Maximalwert scheinen sinnvoll.

Plausibilitätsregeln beachten

Bevor Werte fest in „aux_stats$" eintragen werden, sollten sie nochmals auf Plausibilität geprüft werden: Liegt „SREADTIM" unter 10 ms? Ist „MREADTIM" mindestens um das 1,2-fache größer als „SREADTIM"? Liegt „CPUSPEED" in der Nähe der erwarteten CPU-Taktung? Liegt „MBRC" bei sinnvollen Werten (Werte unter „3" und über „30" sollten misstrauisch machen)? Wer sich mit „MAXTHR" und „SLAVETHR" quält, kann die beiden Werte leer lassen.

Workload-Statistiken manuell in die Tabelle „aux_stats$" eintragen

Die Aktivierung der ermittelten und angepassten Statistikwerte in die Tabelle „aux_stats$" erfolgt schließlich über die Methode „set_system_stats". Abbildung 12 zeigt ein Beispiel.

Bis hierhin sollte klar geworden sein, dass das Ermitteln von realistischen System-Statistiken nicht im Schnellverfahren möglich ist. Deshalb ergibt sich zwangsläufig die Frage, wer diesen Aufwand betreiben sollte. Wer keinerlei Performance-Probleme bemerkt, sollte nicht in Aktionismus verfallen und bei den Defaults bleiben. Wer hingegen bereits die Parameter „optimizer_index_caching", „optimizer_index_cost_adj" oder gar „db_file_multiblock_read_count" angefasst hat, sollte sich mit Workload-Statistiken auseinandersetzen. Wer einmal begonnen hat, Workload-Statistiken zu ermitteln und in die Tabelle „aux_stats$" einzutragen, steht mit diesem Akt in der Pflicht, die Werte auch in gebührenden Abständen zu validieren. Dies ist immer dann ratsam, wenn sich Workload, Hardware oder Daten-Bestand signifikant verändern. Von einem automatischen, regelmäßigen Ak-

Vorschau auf die nächste Ausgabe

Das Schwerpunktthema der Ausgabe 02/2013 lautet

Reporting

Sie erscheint am 12. April 2013

tualisieren der System-Statistiken ohne optische Qualitätskontrolle ist abzuraten, dann verzichtet man besser ganz oder begnügt sich mit NoWorkload-Statistiken. Glücklicherweise scheinen die Rechenmodelle des Optimizers in 11*g* so stabil ausbalanciert und mit hinreichend „IF-Unsinn-THEN-nimm-das"-Anweisungen abgesichert zu sein, dass in den meisten Durchschnitts-Datenbanken unsinnige Einträge in „aux_stats$" unbestraft bleiben.

Fazit

Wer versucht, den 11*g*-Optimizer über Init-Parameter zu beeinflussen, sollte sich mit dem Thema „System-Statistiken" auseinandersetzen. Der Ansatz, dem Optimizer realistische Informationen über die Hardware zu geben, ist ohne Zweifel der richtige. Ernüchternd ist jedoch, wie zeitaufwändig sich das gewissenhafte Ermitteln von Workload-Statistiken gestaltet und wie viel Unsicherheit am Ende trotzdem bleibt. Verwunderlich ist es, dass es keine elegantere Methode für das Ermitteln der sechs Kennwerte geben soll, zumal alle verwendeten Daten im Workload-Repository vorliegen sollten.

Wünschenswert wäre ein Advisor, der automatisch und kontinuierlich sinnvolle Werte ermittelt und vorschlägt. Es bleibt der Eindruck, dass hier eine im Grunde richtige Methodik enthusiastisch begonnen, dann jedoch nur zur mittleren Reife weiterentwickelt wurde. Gelegentlich entsteht darüber hinaus der Eindruck, die Methodik werde stiefmütterlich gepflegt. Der mit 11.2.0.1 eingeführte „SREADTIM/MREADTIM"-Bug 9842771 wurde erst mit Patchset 11.2.0.3 behoben. Das „Read-ahead"-I/O-Verhalten moderner Storage-Systeme kann mit der aktuellen Mess-Methode für „SREADTIM" und „MREADTIM" nicht abgebildet werden. Für zwei der sechs System-Statistiken („MAXTHR" und „SLAVETHR") ist unklar, wie sie in die CBO-Berechnung eingehen. Man fragt sich auch, warum mit „dbms_resource_manager.calibrate_io" eine zusätzliche Benchmark-Methode eingeführt wurde, statt Bestehendes weiterzuentwickeln.

Nicht nachvollziehbar ist die Kurzanleitung, die Oracle in Guides und

```
with aggr as (
    select min(n1)    as minimal
         , max(n1)    as maximal
         , avg(n1)    as mittel
         , stddev(n1) as abweich
      from system.system_stats_twg
     where c4 = 'CPU_SERIO'
       and statid like 'WL_10_ORCL%'
       and n1 > 0          -- Nullen raus
       and n1 <= 10        -- Ausreißer raus
 )
select 'SREADTIM'          as Statistik
     , round(min(n1),2)    as min
     , round(max(n1),2)    as max
     , round(avg(n1),2)    as avg
     , round(stddev(n1),2) as stddev
     , round((stddev(n1)/avg(n1)*100),0) "% STDV"
     , count(*)            as "gült.Mess."
  from system.system_stats_twg
 where c4 = 'CPU_SERIO'
   and statid like 'WL_10_ORCL%'
   and n1 > (select mittel - abweich from aggr)
   and n1 < (select mittel + abweich from aggr)
/

STATISTK  MIN  MAX  AVG STDDEV % STDV gült.Mess.
-------- ---- ---- ---- ------ ------ ----------
SREADTIM 0,57 4,05 2,25    0,9     40        529
```

$$Standard\ Error = \frac{STDDEV}{\sqrt{Messungen}} = \frac{0{,}9}{\sqrt{529}} = \frac{0{,}9}{23} = 0{,}04$$

Abbildung 11: Mathematische Ermittlung von „SREADTIM". Es wird aus allen Werten, die sich innerhalb einer Standardabweichung des vorberechneten gesamten Mittelwerts befinden, ein angepasster Mittelwert errechnet. Grobe Ausreißer (hier „n1= 0" und „n1 > 10") werden gleich zu Beginn eliminiert.

```
begin
 dbms_stats.set_system_stats (pname => 'SREADTIM', pvalue => 2.2);
 dbms_stats.set_system_stats (pname => 'MREADTIM', pvalue => 2.9);
 dbms_stats.set_system_stats (pname => 'CPUSPEED', pvalue => 2400);
 dbms_stats.set_system_stats (pname => 'MBRC',     pvalue => 8);
 dbms_stats.set_system_stats (pname => 'MAXTHR',   pvalue => 100000000);
 dbms_stats.set_system_stats (pname => 'SLAVETHR', pvalue => 2500000);
end;
/
```

Abbildung 12: Manuelles Setzen von Workload-Statistiken. „Tell the optimizer the truth!" – manchmal wünschte man sich, man könne Freitext eingeben, statt nur einer Zahl.

Companion zum Ermitteln von Workload-System-Statistiken an die Hand gibt, als wäre eine schlechte Messung besser als gar keine. Bei der zentralen Bedeutung, die System-Statistiken zukommen soll, wären zeitgemäße Default-Werte sicherlich sinnvoller als zufällige Einzel-Messungen während vermeintlich typischer Workloads.

Thorsten W. Grebe
twgrebe@twg-it.de