Oracle-Performance-Analyse mit frei verfügbaren Tools

Uwe Küchler, OPITZ CONSULTING Deutschland GmbH

Nicht immer stehen einem Datenbank-Administrator GUI-basierte und oftmals sehr teure Tools zur Verfügung, um ein Performance-Problem in der Datenbank zu analysieren. Doch auch mit den vorhandenen Bordmitteln der Datenbank und kostenfrei verfügbaren Tools lässt sich eine gute Diagnose stellen. Dieser Artikel stellt eine Auswahl solcher Werkzeuge vor und legt einen Schwerpunkt auf die Möglichkeiten, die in der Oracle Standard Edition ohne Diagnostics Pack zur Verfügung stehen.

Die Betrachtung der hier vorgestellten Werkzeuge gliedert sich in zwei Anwendungsbereiche:

- Analyse akuter Performance-Probleme
  - Zugriff nur per SSH möglich?
  - Zugriff nur per SQL möglich?
- Historische Analyse ohne AWR und ASH

Wer über die Oracle Enterprise Edition mit Diagnostics Pack und Enterprise Manager verfügt, ist für beide Aufgabenstellungen gut versorgt. Eine grafische Darstellung der Lastdaten, über die ein Drill-Down zu den Verbrauchern von System-Ressourcen gemacht werden kann, erleichtert dem DBA die Arbeit. Was aber, wenn kein Diagnostics Pack lizenziert ist, beispielsweise in der Standard Edition? Dann kommt auch der Einsatz des Enterprise Manager nicht infrage. Es müssen also Alternativen her – möglichst, ohne erst manuell Abfragen unter SQL*Plus absetzen zu müssen. Im Rahmen dieses Artikels kann nur eine sehr kleine Auswahl an verfügbaren Tools vorgestellt werden. Die Wahl fiel auf Tools, die keine Installation von Drittherstellern benötigen, stellt jedoch ansonsten keine Wertung dar.

Kein (X-)Windows? Kein Remote Desktop? Kein Problem!

Aus Sicherheitsgründen sind Datenbank-Server häufig nur per SSH erreichbar. Es kann also nur im Textmodus gearbeitet und keine grafische Oberfläche einge-

setzt werden. Entwicklern oder externen Beratern wird oft sogar der Shell-Zugriff zum Server verwehrt. Dann bleiben nur noch SQL-Abfragen, um sich Informationen über den Zustand der Datenbank zu besorgen. Zunächst der etwas weniger restriktive Fall, bei dem ein SSH-Zugriff erlaubt und bei dem auch eigene Software verwendet werden kann.

AMON – top für die Oracle-Datenbank

AMON wurde von Andrej Simon, einem Mitarbeiter des Oracle-Support-Teams, entwickelt und wird immer noch aktuell gehalten; es ist auf Oracle 10.1 bis 12.2 sowie auf allen Unix- und Linux-Varianten lauffähig. Wer in der Regel über SSH (etwa mittels PuTTY) auf Datenbank-Servern arbeitet, bekommt mit AMON ein Werkzeug an die Hand, das – ähnlich wie das Unix-Tool „top“ auf OS-Ebene – die Last einer Datenbank-Instanz darstellt.

AMON macht nichts anderes, als in regelmäßigen, konfigurierbaren Abständen Performance-Metriken aus dem Data Dictionary der Datenbank auszulesen und die Differenzen zwischen den Intervallen sortiert darzustellen. Im Unterschied zu „top“ sind die Möglichkeiten des Drill-Down allerdings deutlich höher. Ferner zeigt AMON auch diverse Informationen zur Konfiguration der Datenbank an.

AMON muss nicht installiert werden; ein Entpacken des Archivs reicht aus. Es benötigt dann nur noch folgende Umgebungsvariablen:

- ORACLE_HOME
- ORACLE_SID
- LD_LIBRARY_PATH

AMON kann sich damit nach Aufruf sofort an der Datenbank anmelden, was üblicherweise als „/ as sysdba“ erfolgt, allerdings konfigurierbar ist. Damit lässt es sich gut mit personalisierten Accounts oder in Multitenant-Umgebungen einsetzen. Zur Erleichterung des Aufrufs lässt sich mit „alias amon='~/scripts/zk/amon/11.2/amon64_ol5_11r2 -u system -p oracle'“ ein Alias einrichten.

Die folgenden Screenshots zeigen einen typischen Workflow bei der Diagnose eines Performance-Problems. Nehmen wir einmal an, es würde eine auffällig

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Move page forward(f), page back(b)
== General database information. ==

Database character set: AL32UTF8        Force full DB cache: NO
Database log mode     : NOARCHIVELOG    Database role : PRIMARY
Force logging         : NO              Flashback mode: NO
Online redo logs: 3 x 50.0MB
Datafiles: 3.16GB  Tempfiles: 197.0MB

General database initialization parameters.
  cpu_count: 2  db_block_size: 8192  processes: 300

Events and hidden database initialization parameters (V$SYSTEM_PARAMETER).
  Name                    Value
  _ash_size               24M
  _catalog_foreign_restore FALSE

Database components (DBA_REGISTRY).
  Component                          Version     Status
  Oracle Real Application Clusters   12.1.0.2.0  OPTION OFF
  JServer JAVA Virtual Machine       12.1.0.2.0  VALID
  OLAP Analytic Workspace            12.1.0.2.0  VALID
  Oracle Application Express         5.0.3.00.03 VALID
  Oracle Database Catalog Views      12.1.0.2.0  VALID
```

Abbildung 1: Startbildschirm von AMON

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Order by time(1), instance ID(2)
== System time model statistics (V$SYS_TIME_MODEL). ==

Avg. DB time(sec.) per sec. (1, 5, 15 min. ago): 0.24, 0.10, 0.34
Avg. DB wait time ratio (1, 5, 15 min. ago): 8.1%, 15.7%, 14.6%
Avg. DB CPU time ratio (1, 5, 15 min. ago): 91.9%, 84.3%, 85.4%

Time    Statistic
(Sec.)  Name
10.03   DB time
10.03   sql execute elapsed time
9.63    DB CPU
2.69    PL/SQL execution elapsed time
0.75    parse time elapsed
0.40    hard parse elapsed time
0.11    repeated bind elapsed time
```

Abbildung 2: Time Model Statistics

```
$ export LD_LIBRARY_PATH=$ORACLE_HOME/lib:/lib:/usr/lib
$ amon
```

Listing 1

hohe CPU-Last auf dem Server beobachtet und wir möchten feststellen, ob das an der Datenbank liegt und falls ja, woran genau. Starten wir zunächst AMON *(siehe Listing1 und Abbildung 1)*.

AMON empfängt einen mit allgemeinen Informationen zur Datenbank, Hinweisen zu außergewöhnlichen Parametern und gesetzten Events sowie den installierten Komponenten. Das Programm wird über Tastenbefehle gesteuert. Diese können über die Hilfeseiten mit „h“ (alphabetisch) und „hh“ (gruppiert) aufgerufen werden. Dort findet man die Gruppe „Time Model Statistics“ und ruft die erste Seite mit „t“ auf *(siehe Abbildung 2)*.

Nach dem ersten Intervall von zehn Sekunden erscheinen die ersten Werte. Die „Avg CPU Time Ratio“ von 91,9 Prozent zeigt, dass die Instanz vor allem CPU-lastig läuft. Der untere Abschnitt legt mit zehn Sekunden Datenbank-Time in einem Zehn-Sekunden-Intervall nahe, dass hier durchschnittlich eine logische CPU voll ausgelastet läuft. Diese Last wird überwiegend durch SQL-Ausführungen verursacht. Parsing von SQL, das auch sehr CPU-intensiv werden kann, spielt hier mit 0,75 Sekunden nur eine untergeordnete Rolle. Es gilt nun herauszufinden, welche Sessions die momentane Last verursachen. Dazu klickt man ein zweites Mal auf „t“, um die Sessions, sor-

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
Press z(zoom) to get a session details snapshot
== Sess. time model stat. (V$SESS_TIME_MODEL). ==

Avg. DB time(sec.) per sec. (1, 5, 15 min. ago): 1.01, 0.27, 0.41
Avg. DB wait time ratio (1, 5, 15 min. ago): 6.4%, 11.6%, 12.2%
Avg. DB CPU time ratio (1, 5, 15 min. ago): 93.6%, 88.4%, 87.8%

SID,     Con SPID  User   S Program          DB Time DB CPU  BG Time
Ser#     ID                                  (sec.)  (sec.)  (sec.)
40,20750 3   10443 SYSTEM A oracle@vbgeneric 4.00    3.90    0.00
288,7380 3   10465 SYSTEM A oracle@vbgeneric 0.11    0.09    0.00
51,5425  3   6105  SYS    I oracle@vbgeneric 0.00    0.00    0.00
29,24658 3   5798  SYS    I oracle@vbgeneric 0.00    0.00    0.00
```

Abbildung 3: Sessions, sortiert nach Datenbank-Time

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
View session information(1), SQL statement(2), SQL plan(3)
                                        ew. ==
                                  Page: 1/3
SID: 43   Ser: 17530  SPID: 9976  Status: ACTIVE  Cur. Time: 13:04:06

Logon time  : 08-apr-18 12:59:58
Session type: USER  Cont-Name: ORCL (Cont-ID: 3)
DB user name: SYSTEM
OS user name: oracle
SOSID       : 9976  (SPID: 9976  STID: 9976)
Program     : oracle@vbgeneric
Machine     : vbgeneric
Service name: orcl
Client Info :
Module: SQL*Plus
Action:

Event: latch: shared pool
P1,P2,P3: 1611722208, 453, 0
Seconds in wait: 0  Wait time: -1  State: WAITED SHORT TIME
Wait class: Concurrency
```

Abbildung 4: Session-Details

```
   Host: vbgeneric  Instance: orcl12c  PDB: ORCL  User: SYSTEM  Interval: 10 sec
View session information(1), SQL statement(2), SQL plan(3)
== Current SQL statement (V$SQL). - Dynamic view. ==

SID: 40   Ser: 20750  SPID: 10443  Status: ACTIVE

SQL-ID: 82hxvr8kxuzjq  Child number: 0  Plan hash value: 0
SQL Text:
BEGIN dbms_stats.gather_database_stats; END;

Captured bind variables for this SQL cursor (V$SQL_BIND_CAPTURE):

--- No bind data captured.
```

Abbildung 5: Session SQL

tiert nach Datenbank-Time, aufzulisten *(siehe Abbildung 3)*.

In diesem einfachen Beispiel ist genau eine Session mit auffällig hoher DB-Time aufgelistet (schwarz hinterlegt). Für Details zu einer Session kann man in der Liste mit „j“ und „k“ navigieren und dann mit „z“ die Detailseiten aufrufen *(siehe Abbildung 4)*.

Auf dieser Seite stehen zunächst allgemeine Informationen über die Session sowie das zum Zeitpunkt des Snapshots aktive Wait-Event. Die Informationen füllen mehr als eine Bildschirmseite und lassen sich mit „f“ (forward) und „b“ (back) durchblättern. Zur Orientierung lohnt sich ein Blick in die Kopfzeile (schwarz hinterlegt), in der zyklisch Tastenbefehle angezeigt sind, mit denen man von hier aus weitere Details anzeigen kann. In diesem Fall ist es die Taste „2“, die Details zum aktuell ausgeführten SQL in dieser Session liefert *(siehe Abbildung 5)*.

Als Beispiel-Szenario für diesen Artikel wurde in einer separaten SQL*Plus-Session die Berechnung von Datenbank-Statistiken gestartet, um Last zu erzeugen. Damit haben wir an diesem Punkt identifiziert, wer und was die Last auf unserem Server erzeugt. In anderen Fällen könnte man nun noch tiefer forschen und sich beispielsweise den Ausführungsplan für das SQL ansehen (Taste „3“) oder gleich alle Detailinformationen in eine Textdatei schreiben lassen (Taste „0“), um diese dann in Ruhe zu untersuchen oder weiterzuleiten.

Extended SQL Trace und orasrp

Diese Werkzeuge bewegen sich an der Grenze zwischen „nur SSH-Zugriff“ und „nur SQL-Zugriff“. Ein sogenanntes „Extended SQL Trace“ für ein Statement, eine Session oder auch die ganze Datenbank (Vorsicht: gewaltige Datenmengen) können direkt aus SQL heraus ausgelöst werden. Die Herausforderung besteht allerdings darin, die auf dem Datenbank-Server erzeugten Trace-Files auszuwerten oder auf den eigenen Client zu übertragen. Auch der von Oracle mitgelieferte Trace File Profiler „tkprof“ ist oft nur auf dem Server verfügbar. Er ist zwar Teil der Vollinstallation des Oracle-Clients, doch unter Anwendern ist diese immer weniger verbreitet.

Auch das hier vorgestellte „orasrp“ kann Client-seitig laufen (Linux, MacOS, Windows). Es gibt jedoch auch einen interessanten Anwendungsfall für einen Server-seitigen Betrieb: Ein besonderes Feature ist nämlich eine Proxy-Komponente, über die das Tool dann vom Client ferngesteuert werden kann. Wer den Performance-Tuning-Kurs von Oracle besucht hat, wird mit Extended SQL Trace bereits vertraut sein. Für alle anderen hier das Wichtigste dazu in Kürze:

- Wenn die im Data Dictionary vorhandenen Informationen nicht ausreichen, um das Lastverhalten bestimmter SQL-Statements zu untersuchen, kann ein Tracing mit anschließendem Profiling eingesetzt werden.
- Gegenüber dem normalen SQL Trace bietet das Extended SQL Trace zusätzlich noch die Option, jedes einzelne Wait-Event nachzuverfolgen, das während des Tracing auftritt. Damit lassen

```
-- Extended SQL Trace ------------------------------------------------------
-- An-/Abschalten in der eigenen Session
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
SQL> ALTER SESSION SET EVENTS '10046 trace name context off';

-- Auffinden des Tracefiles
SELECT value
FROM   v$diag_info
WHERE  name = 'Default Trace File';

-- An-/Abschalten für eine andere Session
-- (SID und serial# müssen bekannt sein)
SQL> EXEC DBMS_SYSTEM.set_sql_trace_in_session(sid=>123, serial#=>1234, sql_trace=>TRUE);
SQL> EXEC DBMS_SYSTEM.set_sql_trace_in_session(sid=>123, serial#=>1234, sql_trace=>FALSE);

-- An-/Abschalten für eine andere Session
-- (si: Session-ID, se: Serial#, le: Level, 0=aus)
SQL> EXEC DBMS_SYSTEM.set_ev(si=>123, se=>1234, ev=>10046, le=>12, nm=>' ');
SQL> EXEC DBMS_SYSTEM.set_ev(si=>123, se=>1234, ev=>10046, le=>0, nm=>' ');

-- Auffinden des Tracefiles
SELECT p.tracefile
FROM   v$session s
       JOIN v$process p ON s.paddr = p.addr
WHERE  s.sid = 123;
```

Listing 2

sich auch die kleinsten Details einer Session protokollieren.

- Die beim Tracing erzeugten Protokolle sind meist sehr groß und schwer leserlich. Nur in seltenen Fällen wird es nötig sein, den exakten Ablauf in einer Session anhand der Rohdaten nachzuvollziehen.
- In allen anderen Fällen wird ein Profiler eingesetzt, der die Rohdaten gruppiert, sortiert und menschenlesbar aufbereitet. Die Sortierung unterstützt dabei meist das Top-Down-Prinzip, sodass die Top-Ressourcen und Top-SQLs ganz oben sichtbar sind.
- Im Verlaufe eines Tuning-Zyklus kommt ein Tracing wiederholt zum Einsatz, um Vorher-Nachher-Vergleiche anzustellen.

„tkprof", das „Bordmittel" von Oracle, hat allerdings zwei entscheidende Nachteile:

- Es kann mit den Zusatz-Informationen des Extended SQL Trace nichts anfangen.
- Es extrahiert die SQL-Ausführungspläne nicht aus dem Trace-File, sondern erzeugt sie zur Laufzeit. Die so generierten Ausführungspläne können daher von denen im Trace-File abweichen.

Besser geeignet sind hier zwei neuere Tools, die auch mit den erweiterten Informationen umgehen können. Eines davon ist der Trace Analyzer (TRCA) von Oracle, der unter der MOS Doc ID 224270.1 erhältlich ist. Das zweite Tool, das hier infrage kommt, nennt sich „orasrp". Es muss gegenüber dem TRCA keine Objekte in der Datenbank installieren und erfordert auch keine Verbindung zur Datenbank. Das macht es besonders für Ad-hoc- und Remote-Untersuchungen praktikabel.

„orasrp" wurde vom russischen Datenbank-Administrator Egor Starostin entwickelt. Das letzte Update der Software stammt vom April 2013, es ist also deutlich älter als das Release von Oracle 12.2. Allerdings können Trace-Files von Oracle 12.2 damit problemlos verarbeitet werden. *Listing 2* zeigt die Erzeugung eines Extended SQL Trace:

Nachdem das Trace File erzeugt und der Dateiname bekannt ist, kann „orasrp" mit „$ orasrp --google-charts tracefile.trc parsedfile.html" seine Arbeit beginnen. Die Anweisung „--google-charts" ist optional und erzeugt zusätzlichen Code für die Generierung einer Tortengrafik mithilfe von Google Charts. Nach ein paar Sekunden ist die Ausgabe in die HTML-Datei, die in der Kommandozeile benannt wird, fertig und die Datei kann in einem beliebigen Browser geöffnet werden *(siehe Abbildung 6)*.

Das Beispiel stammt aus einer realen Produktionsumgebung. Diese wurde zwar kurz zuvor von Oracle 11.2 nach 12.1 migriert, doch hatten sich danach die Laufzeiten verschiedener Batch-Jobs um mehr als den Faktor zwei verschlechtert – obwohl eine Verbesserung zu erwarten war, weil im Zuge der Migration sogar auf eine leistungsfähigere Hardware gewechselt wurde. Da ein erster Vergleich der SQL-Ausführungspläne keine Unterschiede zwischen dem alten und dem neuen System zeigte, fiel die Entscheidung, ein Extended SQL Trace speziell für die Batch-Session durchzuführen.

Im „Session Flat Profile" fällt auf, dass „db file sequential read" den größten Teil der Last ausmacht (also Random I/O von einzelnen Datenblöcken) und die maximale Wartezeit für dieses Event mit 1,65 Sekunden viel zu groß ist. Es geht bei diesem Event um einzelne Block-Zugriffe; wenn dabei auf eine Festplatte zugegriffen werden muss, dann sind bei heutigen Festplatten Zugriffszeiten von vier bis acht Millisekunden üblich (siehe *Abbildung 6*, gelbe Hervorhebung). Da „orasrp" die einzelnen Abschnitte untereinander verlinkt, springt man mit einem Klick auf das Event „db file sequen-

tial read“ zu den „Top 5 Statements per Event“ *(siehe Abbildung 7)*.

Hier fällt auf, dass das Top-SQL Nr. 1 bereits für 70 Prozent der Events verantwortlich ist. Darüber hinaus taucht bei diesem SQL wieder die lange, maximale Wartezeit von 1,65 Sekunden auf. Es gilt jetzt, dieses SQL näher zu betrachten; dazu ein Klick auf dessen Hash-Wert in der Tabelle *(siehe Abbildung 8)*.

Der Screenshot zeigt neben dem Ausführungsplan und den kumulativen Statistiken, wie sie auch von „tkprof“ her bekannt sind, ein Wait-Event-Profil, spezifisch für dieses SQL. Auch hier trägt das

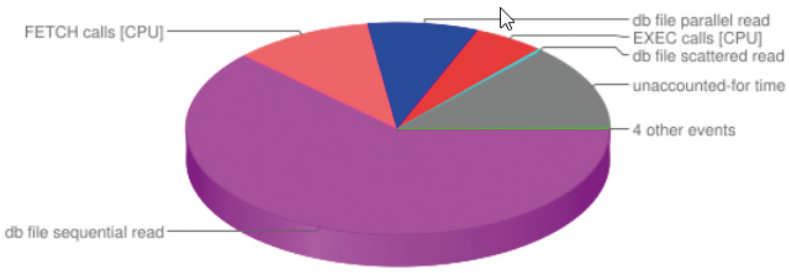
Session Summary

Instance Name: xdb
Node Name: oraxdb01
Oracle Release: 12.1.0.2.0
Session ID: 12.56325
Total SQL Statements: 37 (23 user statements, 14 internal statements)
Total Transactions: 4 (0 rollback(s), 1 read-only)

Report Shortcuts

Session Flat Profile
Top 5 Statements per Event
Session Call Graph
Events Histograms
Statements

Session Flat Profile Pie Chart



Session Flat Profile

Event Name	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
db file sequential read	84.2%	717.3377s	106,623	0.0067s	0.0001s	1.6540s
FETCH calls [CPU]	14.4%	122.5760s	186,707	0.0007s	0.0000s	0.6675s
db file parallel read	11.4%	96.6988s	3,605	0.0268s	0.0004s	0.8373s
EXEC calls [CPU]	7.3%	61.9997s	220,844	0.0003s	0.0000s	33.2023s
db file scattered read	0.4%	3.6598s	265	0.0138s	0.0003s	0.2412s
log file switch completion	0.0%	0.0798s	1	0.0798s	0.0798s	0.0798s
control file sequential read	0.0%	0.0012s	7	0.0002s	0.0001s	0.0003s
log file sync	0.0%	0.0003s	1	0.0003s	0.0003s	0.0003s
PARSE calls [CPU]	0.0%	0.0000s	35	0.0000s	0.0000s	0.0000s
unaccounted-for time	-17.7%	-150.4595s				
Total	**100.0%**	**851.8937s**				

Abbildung 6: Lastprofil der Session

Top 5 Statements per Event

db file sequential read

SQL Hash Value	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
3804044889	70.0%	502.0518s	48,650	0.0103s	0.0001s	1.6540s
1980990944	18.0%	129.3922s	30,672	0.0042s	0.0001s	0.2315s
1428548520	11.1%	79.3675s	24,182	0.0033s	0.0001s	0.4565s
2303779568	0.8%	6.0044s	2,642	0.0023s	0.0001s	0.1868s
3629108195	0.0%	0.2941s	28	0.0105s	0.0001s	0.0592s
3 others	0.0%	0.2276s	449	0.0005s	0.0001s	0.1164s
Total	**100.0%**	**717.3377s**	**106,623**	**0.0067s**	**0.0001s**	**1.6540s**

Abbildung 7: Top-SQL pro Event

SQL Statements

SQL Hash Value: 3804044889 SQL Id: czv8xxzjbu7kt uid: 73 depth: 2 optimizer mode: ALL_ROWS

Statement Text

```
SELECT *
FROM PERS_HISTORIE
WHERE PSN_ID = :B3 AND VORDAT = 'J' AND TO_CHAR(GUELTIG_AB,'YYYYMM') = :B2 ||
  :B1 AND ( UPPER(HIST_FELD) = 'BEITRAG' OR UPPER(HIST_FELD) = 'MTGLART' OR
  UPPER(HIST_FELD) = 'FKEITTG' OR UPPER(HIST_FELD) = 'ZPANFMN' OR
  UPPER(HIST_FELD) = 'REGSPN' OR UPPER(HIST_FELD) = 'ZPDAUER' OR
  UPPER(HIST_FELD) = 'ZAHLART' )
ORDER BY HIST_FELD, AEND_DAT
```

Statement Cumulative Statistics

Call	Cache Misses	Count	- Seconds -		Physical Reads	- Logical Reads -		Rows
			CPU	Elapsed		Consistent	Current	
Parse	0	0	0.0000s	0.0000s	0	0	0	0
Exec	0	24,085	0.8500s	1.3822s	0	0	0	0
Fetch		24,121	87.6810s	605.1096s	58,291	283,917	0	36
Total	**0**	**48,206**	**88.5310s**	**606.4918s**	**58,291**	**283,917**	**0**	**36**
Per Fch	0.0	2.0	0.0037s	0.0251s	2.4	11.8	0.0	0.0
Per Row	0.0	1,339.1	2.4592s	16.8470s	1,619.2	7,886.6	0.0	1.0

Statement Flat Profile

Event Name	% Time	Seconds	Calls	- Time per Call -		
				Avg	Min	Max
db file sequential read	72.7%	502.0518s	48,650	0.0103s	0.0001s	1.6540s
db file parallel read	14.0%	96.6988s	3,605	0.0268s	0.0004s	0.8373s
FETCH calls [CPU]	12.7%	87.6810s	24,121	0.0036s	0.0000s	0.6675s
db file scattered read	0.5%	3.6598s	265	0.0138s	0.0003s	0.2412s
EXEC calls [CPU]	0.1%	0.8500s	24,085	0.0000s	0.0000s	0.0059s
Total	**100.0%**	**690.9415s**				

Abbildung 8: SQL-Details (Ausschnitt)

Event „db file sequential read“ mit 72 Prozent am meisten zur Laufzeit bei. Auffällig ist auch die doch ziemlich hohe durchschnittliche Wartezeit mit mehr als zehn Millisekunden.

Die in diesem Beispiel gesammelten Informationen wurden im Rahmen eines Support Request mit Oracle Support besprochen. In diesem Fall kam heraus, dass es sich hier tatsächlich um ein Problem beim Filesystem-I/O handelt.

Historische Analyse ohne AWR

Auch für eine historische Analyse gibt es verschiedene Angebote, die vor allem auf eine Nachbildung der Active Session History (ASH) setzen. Da diese aber in der Regel eine Installation eigener Objekte in der Datenbank erfordern und dies oftmals als potenzielles Sicherheitsrisiko angesehen wird, soll hier auf ein herstellereigenes Tool von Oracle eingegangen werden: Statspack.

Oracle aktualisiert Statspack nach wie vor mit jedem Release, einschließlich Version 18c. Allerdings tauchten in den letzten Releases immer wieder Fehler auf, die nicht im Rahmen von Patch-Sets behoben wurden. Die zwei wesentlichen Bugs sind:

- Die Klassifizierung der Wait-Events wird in Statspack separat vom Data Dictionary vorgehalten, aber oft nicht aktualisiert. Dadurch erscheinen weniger relevante Events, sogenannte „Idle Events“, in den Top-Events plötzlich ganz oben und verschleiern damit die Sicht auf die wirklich relevanten Events.
- Die Tabelle für historisierte SQL-Ausführungspläne ist nicht mehr identisch mit der Tabelle „PLAN_TABLE“ im Data Dictionary. Es fehlt die Spalte „TIMESTAMP“; das stört allerdings nur, wenn man das Package „DBMS_XPLAN“ mit einem mit Statspack historisierten Plan verwenden möchte.

Auf GitHub gibt es im „SQL-Zauberkasten“ einen Skript-Satz, mit dem sowohl die manuellen Installationsschritte als auch der Workaround für die Idle Events automatisiert durchgeführt werden können *(siehe „https://github.com/Rendanic/SQL-Zauberkasten/tree/master/sql/statspack/spcreate“)*. Für die fehlende Spalte gibt das Skript bei Bedarf ein SQL-Statement aus. Die Installation vereinfacht sich daher zu folgenden Schritten:

- Tablespace anlegen
- Das Skript „statspack_configuration.sql“ auf die gewünschten Werte für Passwort, Tablespaces, Vorhaltedauer, Snapshot-Intervall und Detailtiefe anpassen
- Das Skript „statspack_create.sql“ ausführen

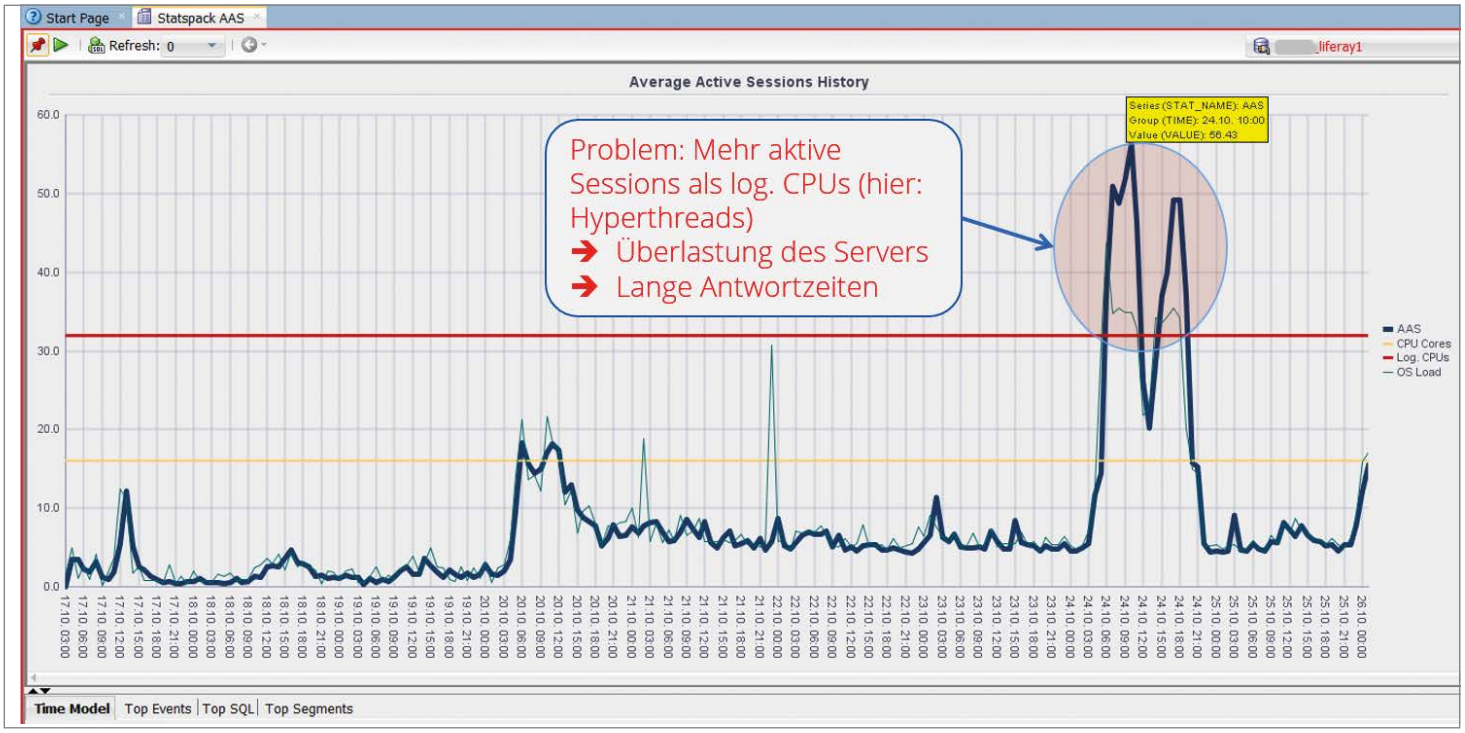
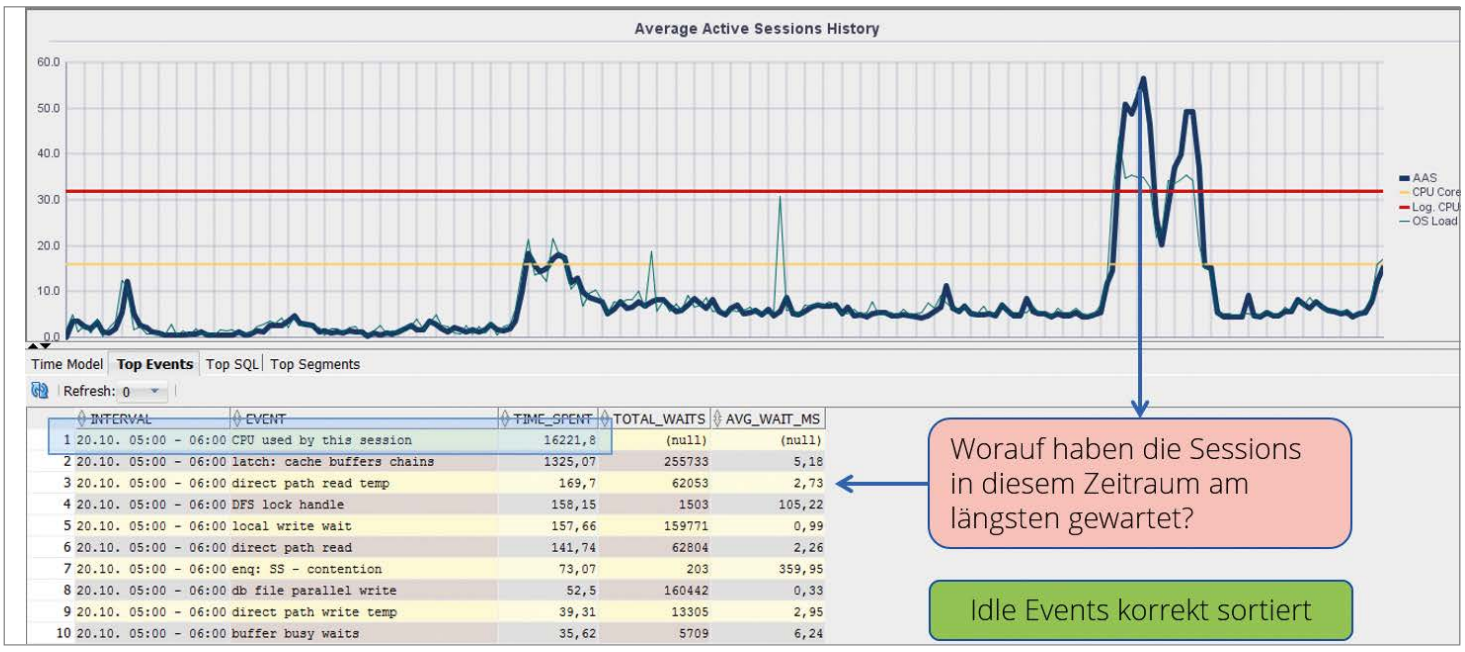


Abbildung 9: Average Active Sessions (AAS)

Nach der Ausführung sind die Jobs sofort aktiv und erstellen im gewünschten Rhythmus Snapshots von Performance-Metriken und ausgeführtem SQL. Die Auswertung der historisierten Daten kann, sobald mindestens zwei Snapshots vorliegen, mithilfe des SQL-Skripts „spreport.sql" erfolgen, das unter „$ORACLE_HOME/rdbms/admin" vorliegt. Dort steht in der Datei „spdoc.txt" auch eine kurze Dokumentation zur Verfügung.

Wer mehr visuell veranlagt ist, bekommt mit dem Oracle SQL Developer ein weiteres kostenfreies Werkzeug an die Hand, mit dem sich die in den Statspack-Tabellen liegenden Daten grafisch aufbereiten lassen. Dazu gibt es, ebenfalls auf GitHub, fertige Reports zum Herunterladen und Importieren. *Abbildung 9* zeigt, wie so ein Report aussehen kann, der von einem produktiven System stammt.

Der erste Teil zeigt einen Graphen über die Historie von durchschnittlich aktiven Sessions im Zeitraum zwischen den einzelnen Snapshots. Falls verfügbar, wird die Last auf dem Server als dünne Linie dargestellt. Insbesondere bei mehreren Datenbank-Instanzen auf einem Server ist das ein wichtiger Vergleichswert, da die untersuchte Datenbank vielleicht gar nicht für die Serverlast



	INTERVAL	EVENT	TIME_SPENT	TOTAL_WAITS	AVG_WAIT_MS
1	20.10. 05:00 - 06:00	CPU used by this session	16221,8	(null)	(null)
2	20.10. 05:00 - 06:00	latch: cache buffers chains	1325,07	255733	5,18
3	20.10. 05:00 - 06:00	direct path read temp	169,7	62053	2,73
4	20.10. 05:00 - 06:00	DFS lock handle	158,15	1503	105,22
5	20.10. 05:00 - 06:00	local write wait	157,66	159771	0,99
6	20.10. 05:00 - 06:00	direct path read	141,74	62804	2,26
7	20.10. 05:00 - 06:00	enq: SS - contention	73,07	203	359,95
8	20.10. 05:00 - 06:00	db file parallel write	52,5	160442	0,33
9	20.10. 05:00 - 06:00	direct path write temp	39,31	13305	2,95
10	20.10. 05:00 - 06:00	buffer busy waits	35,62	5709	6,24

Abbildung 10: Top-Wait-Events

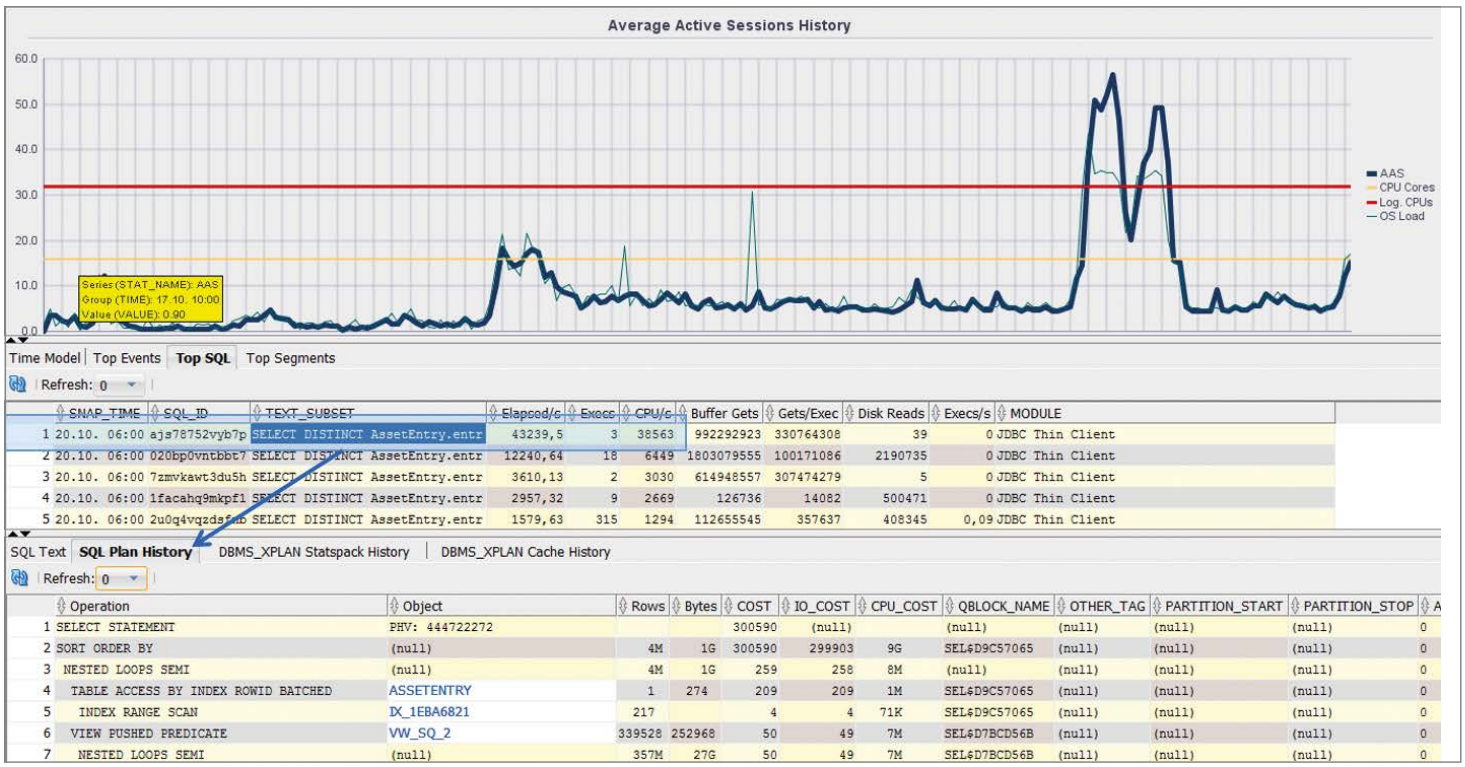


Abbildung 11: Top-SQL mit historisierten Ausführungsplänen

verantwortlich ist. Im hervorgehobenen Zeitraum korreliert die Serverlast jedoch mit den AAS. Beide Metriken überschreiten die Anzahl der logischen CPUs (auf diesem System sind das Hyperthreads) deutlich; hier liegt offenbar eine Überlastung vor. Beim Klick auf einen Punkt im Graphen wird das erste Detailfenster befüllt *(siehe Abbildung 10)*.

Unter dem Reiter „Top Events" finden sich alle zwischen dem gewählten und dem vorherigen Snapshot aufgetretenen Wait-Events. Das SQL hinter dem Report berücksichtigt bereits die oben erwähnte Problematik mit den Idle Events und sortiert diese alle nach unten. Will man nun herausfinden, welches das Top-SQL im betrachteten Zeitraum war, wechselt man zum entsprechenden Reiter und bekommt daraufhin eine zusätzliche Detailstufe angezeigt *(siehe Abbildung 11)*.

Das Top-SQL wird zunächst nach Gesamtlaufzeit absteigend sortiert. Eine Sortierung nach anderen Spalten ist im SQL Developer durch Anklicken möglich.

Wenn ein SQL angewählt wird (durch einen Klick irgendwo innerhalb der gewünschten Zeile), werden die Details dazu im unteren Teilfenster angezeigt. In den historisierten Ausführungsplänen ist es darüber hinaus möglich, Einträge in der Spalte „Object" (die blau auf weiß hervorgehobenen Namen von Tabellen, Views und Indizes) anzuklicken und damit die Detailfenster zu diesen Objekten zu öffnen.

Fazit

Wenn es an die Diagnose von Performance-Problemen geht, kann mit Bordmitteln der Oracle Standard Edition und hilfreichen, kostenfreien Tools von Drittanbietern oder von Oracle Support auch ohne teure Optionen eine zügige Analyse durchgeführt werden.

Referenzen

- AMON: *https://sites.google.com/site/freetoolamon*
- Erzeugen und Auswerten von Extended SQL Trace: *https://oracle-base.com/articles/misc/sql-trace-10046-trcsess-and-tkprof*
- „orasrp"-Anleitung und Download: *http://oracledba.ru/orasrp*
- SQL_TRACE/Event 10046 Trace File Analyzer (trca): MOS Doc ID 224270.1
- Hilfsskripte zur Installation von Statspack: *https://github.com/Rendanic/SQL-Zauberkasten/tree/master/sql/statspack/spcreate*
- Performance-Reports für den SQL Developer: *https://github.com/oraculix/sql-developer-tools/tree/master/reports*, hier vorgestellt: „Statspack_AAS_3Level.xml"

Uwe Küchler
uwe.kuechler@opitz-consulting.com