Advanced Oracle Troubleshooting – Live Session

Randolf Geist
Freelance Consultant
Mannheim, Germany

Schlüsselworte:

Advanced Oracle Troubleshooting, Live Session, Performance

Einleitung:

In this session we'll walk through some common troubleshooting scenarios, starting with the usual inefficient execution plan issues, showing that the Oracle Wait Interface and Session Statistics are usually well suited to spot anomalies and determine where excess work happens.

We'll then continue with some less common scenarios where the wait Interface or session statistics tend to be useless and demonstrate some advanced troubleshooting techniques that still allow a systematic approach under such circumstances.

This is a live demonstration session along with a couple of slides that describe the techniques applied.

The following will be covered during this session:

- SQL Monitoring
- Extended SQL trace resp. Oracle Wait Interface (I/O, contention, locks)
- Session Statistics
- Active Session History / S-ASH
- Latch/Mutex Activity
- Detailed Consistent gets analysis (excess consistent gets)
- Oracle heapdump analysis
- Operating System level process tracing

Basic Troubleshooting Techniques

Before we get too excited about Advanced Troubleshooting it is important to remember that most performance issues can be identified using basic troubleshooting techniques.

So what are your most important basic troubleshooting tools?

- Systematic approach
    - Always focus on the most important and affected business process; if possible avoid troubleshooting by looking at the "system" from an aggregated view
- Once you have identified these processes the most important things to check first are:

    - Session Statistics and Wait Interface => Best tool: Tanel Poder's "snapper". Note that snapper covers more than just statistics and wait interface information
- If you have identified SQLs or sessions that are taking longer than expected, use proven technology: Extended SQL trace along with DBMS_XPLAN.DISPLAY_CURSOR / STATISTICS_LEVEL = ALL (10g and later)
    - These two tools are complementary: Both tell you where most of the time is spent. Extended SQL trace with waits enabled shows additionally the wait events whereas DBMS_XPLAN.DISPLAY_CURSOR shows the estimated cardinalities together with the actual ones. Furthermore it shows the estimated memory requirements along with the actual memory used and the amount of TEMP space used when spilling to disk
- Bad execution plans are often caused by incorrect cardinality estimates - check the estimated and actual cardinalities reported by DBMS_XPLAN.DISPLAY_CURSOR
- In 11g SQL Real-Time Monitoring has been added - however this requires a tuning pack & diagnostics license
    - V$SQL_MONITOR
    - V$SQL_PLAN_MONITOR
    - DBMS_SQLTUNE.REPORT_SQL_MONITOR
      Combines information from V$SQL_PLAN_MONITOR, V$SESSION_LONGOPS and V$ACTIVE_SESSION_HISTORY
      Loads of parameters and options - worth to study the details
- Pimp up your basic tools:
    - Use Adrian Billington's XPLAN wrapper for DBMS_XPLAN.DISPLAY* functions to add execution order and parent operation info
    - Alberto Dell'Era's XPLAN / XTRACE tools (http://www.adellera.it/)
    - Use alternative trace file analyzers in addition to TKPROF:
        - Oracle's own Trace Analyzer (TRCA / TRCANLZR) See MOS Note 224270.1, and see also extended Explain Plan 215187.1 SQLT (SQLTXPLAIN)
          Both tools require an installation and are a bit cumbersome to use, however they provide a wealth of detailed information
        - Christian Antognini's TVD$XTAT (http://antognini.ch/top)
        - OraSRP (http://www.oradba.ru/orasrp)
    - Support your developers by using Method R's MR Trace SQLDeveloper plug-in - allows to automatically fetching trace files from the server or write your own little helper tool that transports trace files to the client automatically

Live Demo

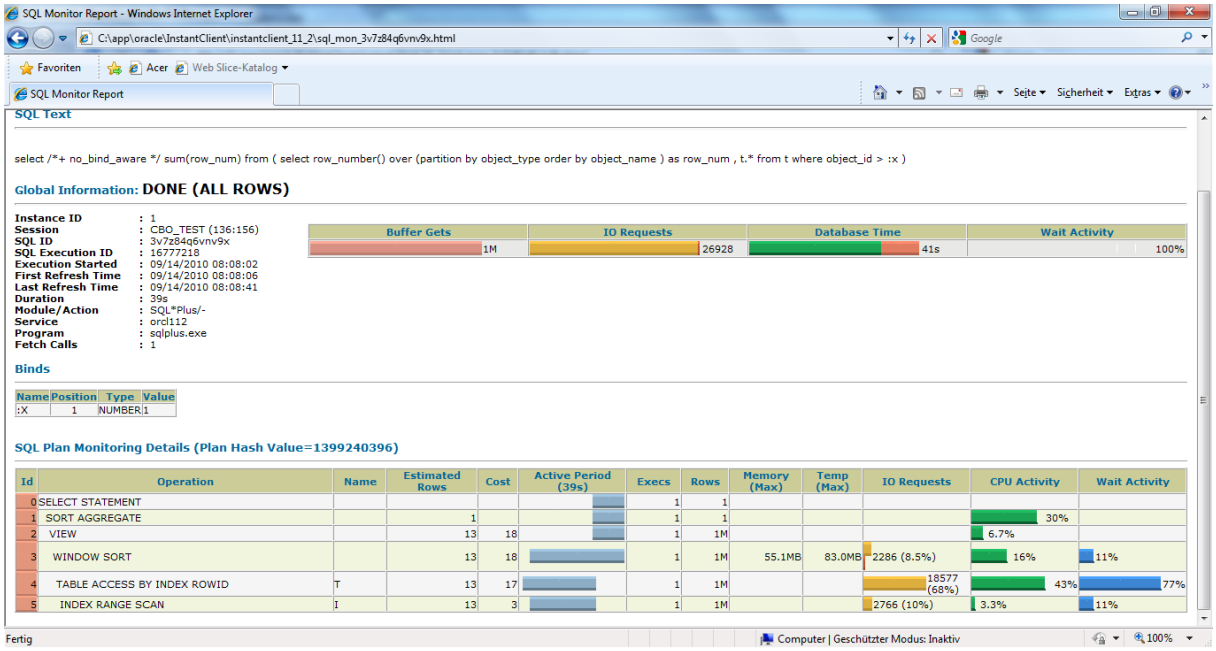

Image 1: Real-Time SQL Monitoring

Active Session History (ASH)

Another important tool for troubleshooting is the so called “Active Session History”. This feature has officially been added to Oracle in version 10g, but requires an additional license (diagnostic & tuning pack).

Sometimes you get a call "My process was slow 10 minutes ago, but it's completed now"

- Oracle 10g has added Active Session History (ASH)
  - Interfaces: V$ACTIVE_SESSION_HISTORY and DBA_HIST_ACTIVE_SESS_HISTORY
  - Standard report available: RDBMS/ADMIN/ASHRPT.SQL
  - Requires additional license diagnostic & tuning pack
- The good news is that Active Session History is not rocket science, in principle it is about sampling V$SESSION (or its underlying structures). Therefore you don't need necessarily an additional license, but can do it yourself or use instead Kyle Hailey's S-ASH scripts for an "Active Session History" without ASH license. See http://www.ashmasters.com or the OraSASH homepage on Sourceforge.net.
- When using connection pooling good instrumentation using client identifiers is important for effective troubleshooting

Live Demo

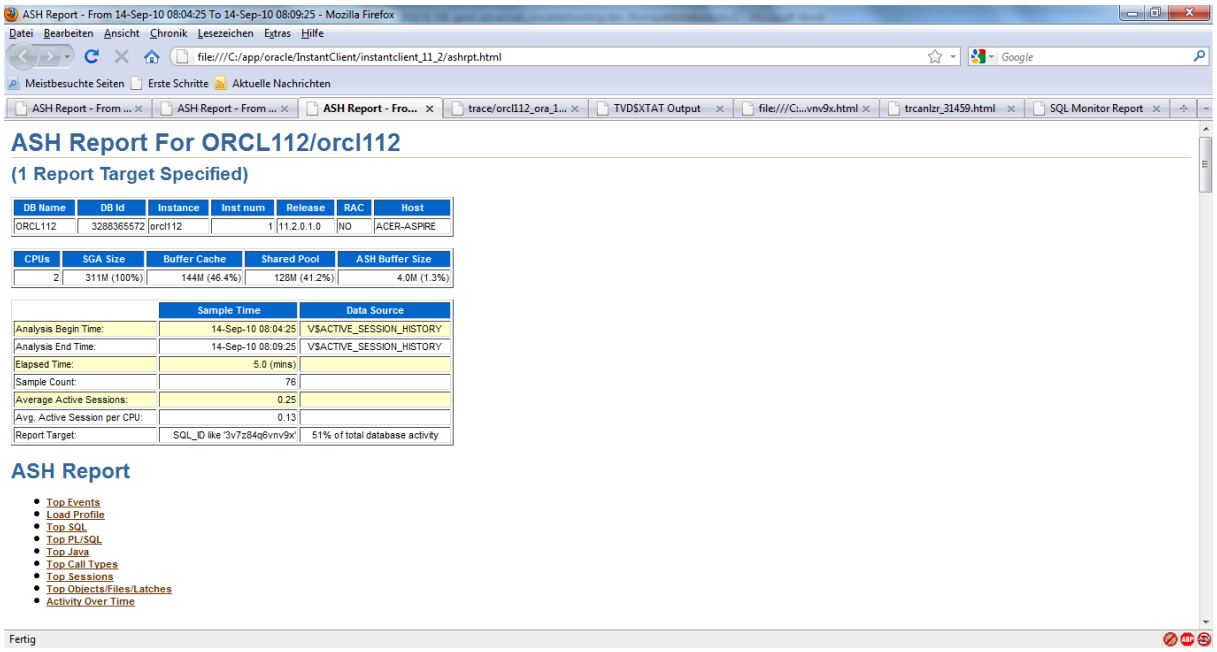

Image 2: ASH Report

Latch/Mutex Activity

- If multiple processes compete for certain resources you might see so called "latch contention"
- Latches and Mutexes protect and serialize access to in-memory structures of the SGA
- Most commonly "latch cache buffer chain" contention when multiple processes excessively attempt to access buffers protected by the same latch buffer chain
- Another common issue is library cache contention due to excessive hard parsing activity
- Note that latch contention usually is a symptom rather than a cause
- Since spinning on latches burns CPU, excessive buffer gets can significantly increase CPU load
- On the other hand if you overload your CPUs then latch contention can be exaggerated by this CPU overload since the latches are held for too long and therefore contention of latches will be observed
- So latch contention can cause CPU load but can also be a symptom of CPU overload
- In general there is not much you can do about this except for changing the application logic
- In case of excessive buffer gets check if a bad statement execution plan causes this (mostly plans including NESTED LOOP operations), or in case of "hot blocks" determine the cause, for instance a sequence-based insert into an indexed column might cause such issues - then a remedy might be to modify the index to reverse or global hash partitioned
- Details about latch/mutex activity can be obtained from dynamic performance views and internal X$ fixed tables
- Tanel Poder's tool set comes handy: LATCHPROF / LATCHPROFX (only as SYS or when X$ fixed views are accessible) and MUTEXPROF
- More information: Tanel Poder's presentation and seminars about Latch / Mutex troubleshooting

Live Demo

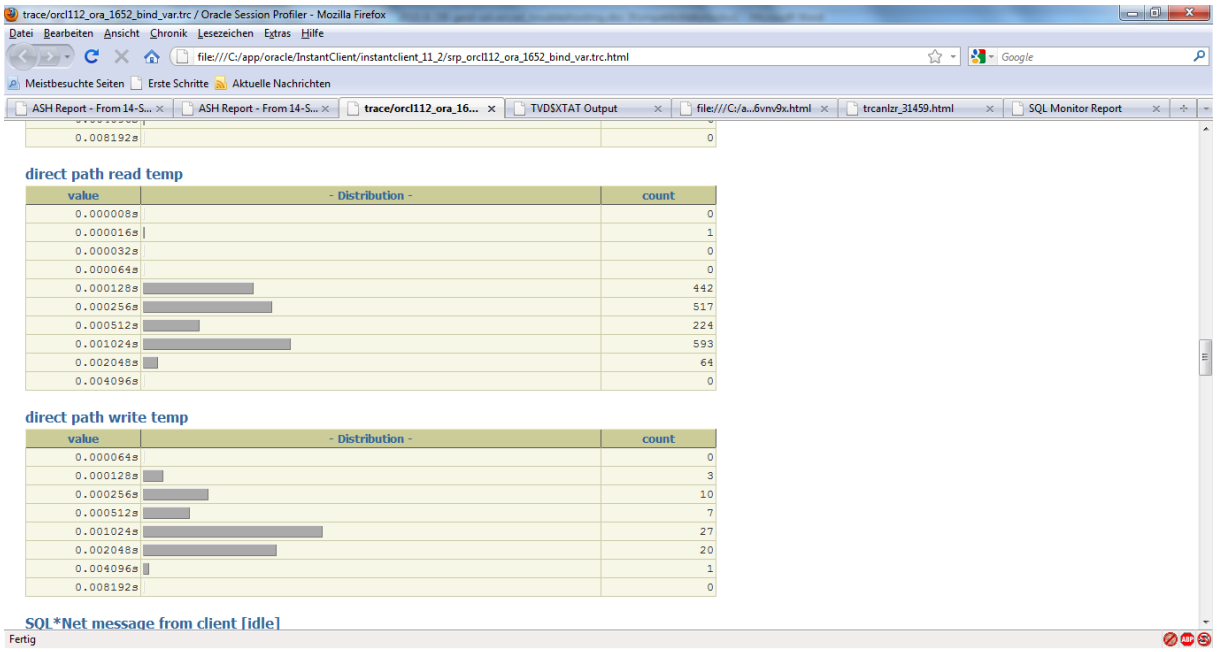

Image 3: Extended SQL Trace file analysis using OraSRP

Detailed Consistent gets analysis (excess consistent gets)

Tracing and DBMS_XPLAN.DISPLAY_CURSOR tell you the number of consistent gets performed, but don't tell you the reason why they have been performed. If you suspect that the number of consistent gets is higher than expected, it would be interesting to know the "reason" why the consistent get was performed.

- Excessive consistent gets can be caused by different reasons
  - The most obvious reason: Reconstructing older versions of a block by applying undo
  - Bugs: many ASSM related problems in past versions
- Up to version 10g internal X$ tables can be used to obtain information about the reason, for 11g the internal structure has changed the contents are not updated any longer
- If you connect with a user capable of accessing X$ tables, snapper will show you this as well
- Jonathan Lewis provides a nice script which takes a snapshot of this structure and outputs the delta afterwards, see http://www.jlcomp.demon.co.uk/buffer_usage.html
- Note that this is global information and not limited to particular sessions, so on-going activity may influence the measurement
- Alternatively use event 10200/10202 to trace consistent gets

Live Demo

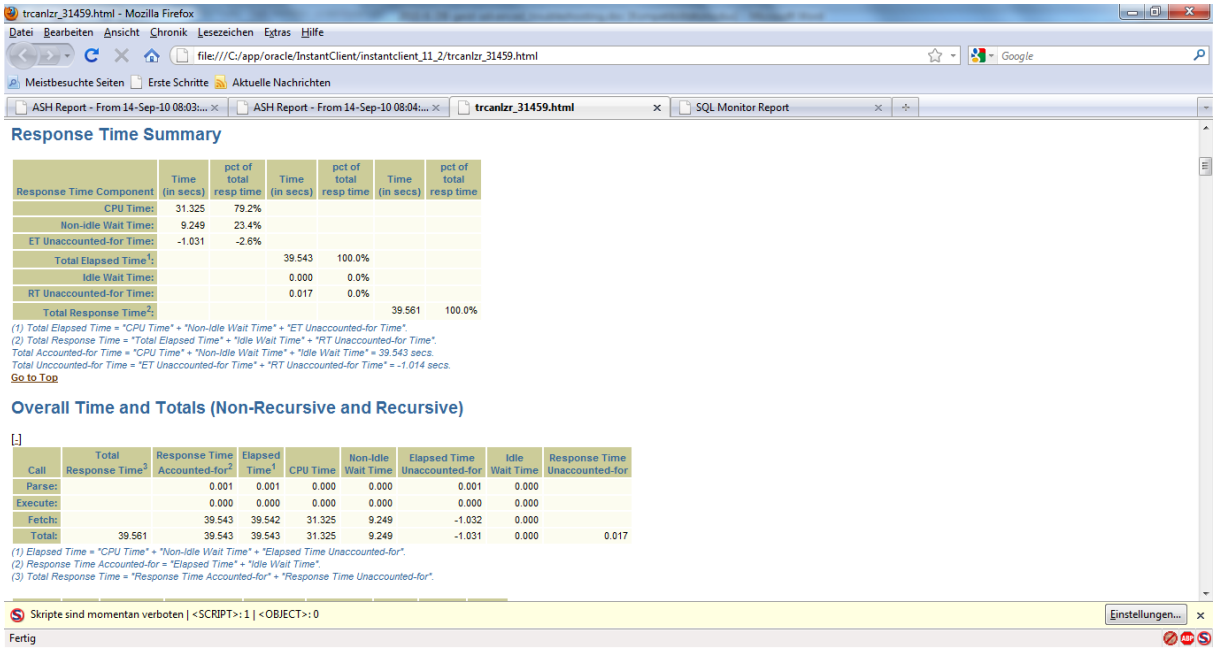

Image 4: Oracle Trace Analyzer output

When basic troubleshooting techniques are not sufficient

Now we'll come to the point where above tools will not allow a detailed diagnosis any longer.

You hit the limits for example if you have a case that:

- Consistently sits on CPU, but doesn't show anything in the session statistics, so you don't have a clue what the process is actually doing
- Performs some uninstrumented code section and therefore doesn't update the wait interface and/or the session statistics accordingly

Examples:

- PL/SQL processing
- Parsing bugs
- Instrumentation bugs: External table access up to 11g

In these cases a different approach is required if we need more details

Trace the process on OS level using:

- Tanel Poder's OSStackProf
- DTrace (Solaris)
- procstack (AIX)
- pstack / gdb backtrace (Linux)
- In future may be: probevue (>= AIX 6.1), systemtap (Linux)

Sometimes it is also useful to find out the system calls performed by the process using:

- truss (Linux, AIX)
- strace
- ptrace

Use Metalink / MOS document 175982.1 to get an idea about the purpose of the functions traced

Live Demo

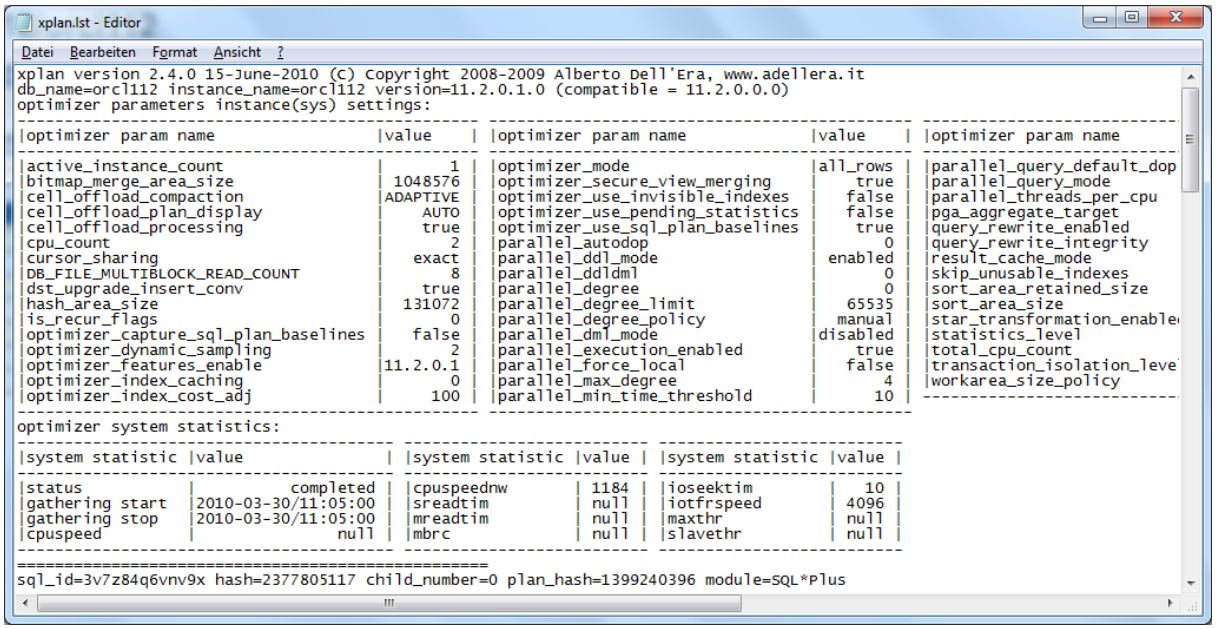

Image 5: Alberto Dell'Era's XPLAN tool

OS Explain Plan

The same technique can be used to obtain information about a currently executing execution plan, since the row sources used to execute a SQL statement are represented by corresponding C functions in the Oracle code.

Live Demo

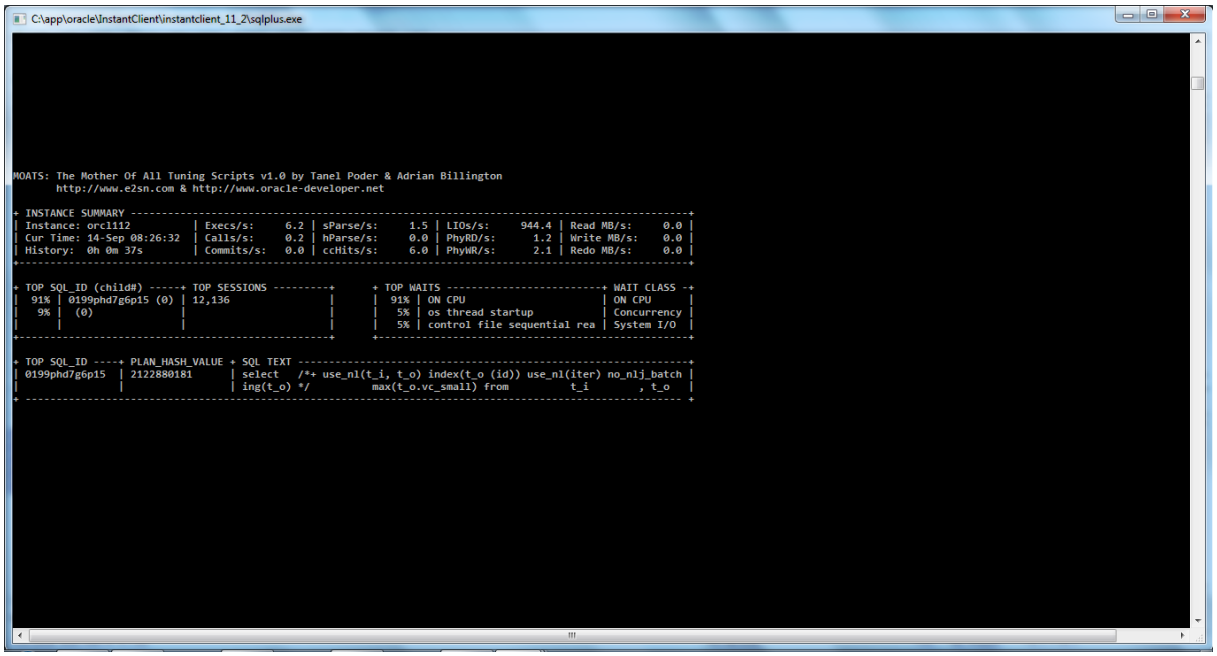

Image 6: Tanel Poder's MOATS tool

Other advanced techniques – heapdumps,errorstack dumps, systemstate dumps etc.

If you can't tell from the call stack what is going on, sometimes it might be helpful to analyze the memory using heapdumps, in particular if you see suspicious activity in the session statistics (for example, ever increasing PGA / SGA memory consumption)

Live Demo

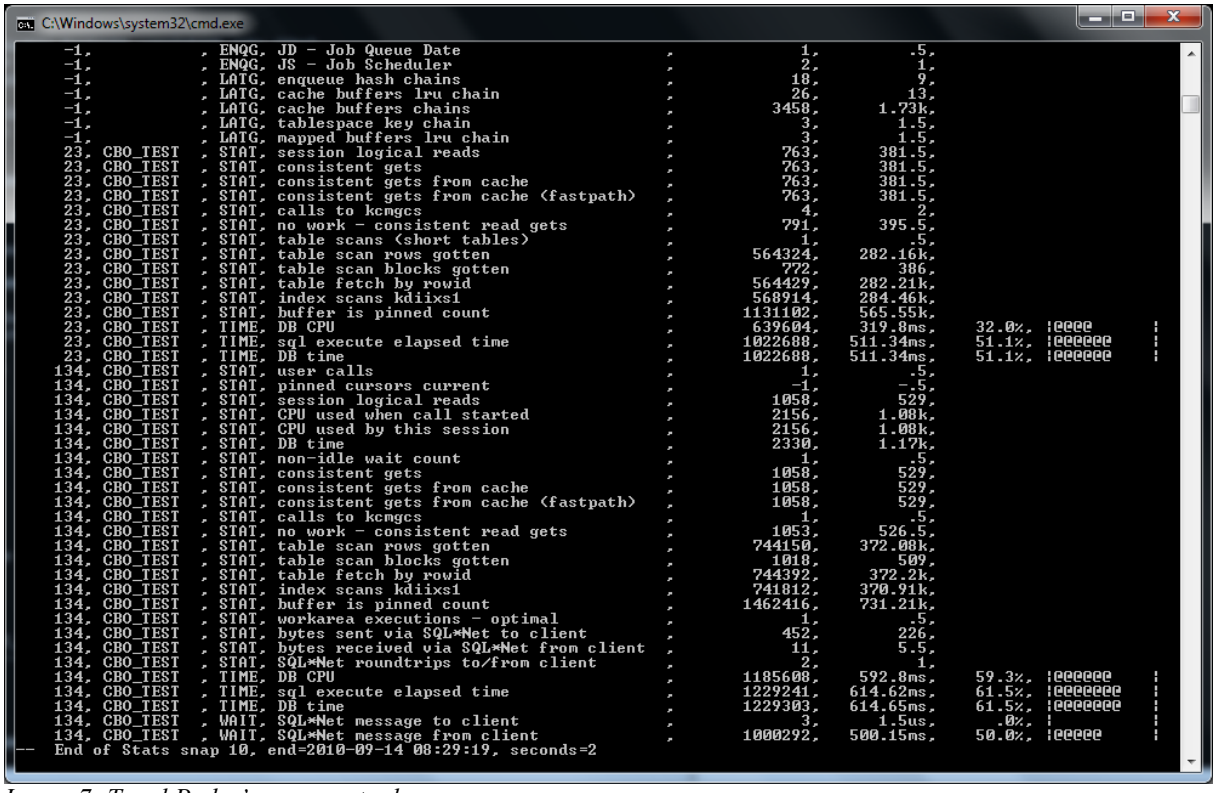

Image 7: Tanel Poder's snapper tool

Kontaktadresse:

Randolf Geist
Freelance Consultant
Ersteiner Straße 15
D-68229 Mannheim

Telefon: +49 (0) 170-758 1171
E-Mail info@sqltools-plusplus.org
Internet: http://oracle-randolf.blogspot.com