Oracle System Performance Analysis Using Oracle Event 10046

Cary V. Millsap

Hotsos Enterprises, Ltd.

Jeffrey L. Holt

Hotsos Enterprises, Ltd.

Revision History		
Revision 1.2 (2d beta draft)	18 March 2002	cvm
Revision 1.1 (beta draft)	14 March 2002	cvm
Revision 1.0 (alpha draft)	13 February 2002	cvm

Table of Contents

1. Executive Summary

In more than a hundred performance improvement opportunities, Hotsos staff have taught customers how to resolve system performance problems that had eluded solution for months or years. The whole process—including data collection, analysis, research, and repair—typically consumes no more than a few hours. The new method is based upon Oracle's ability to instrument its own performance with a standard Oracle product feature

that has been available since release 7.0.12 (circa 1992). This document will help you understand the value of the detailed SQL statement timing data produced by Oracle's so-called *event 10046*. The paper also details how to activate event 10046 data collection.

2. Introduction to Oracle Event 10046

The Oracle database kernel is instrumented with over four hundred so-called "pseudo-error debugging events." [1] One of the most important of these events for the system performance analyst is event 10046, "enable SQL statement timing." Activating this event for an Oracle session instructs the Oracle kernel to print detailed timing information for that session to an Oracle trace file. A sample of 10046 data is provided later in this document.

Event 10046 is an attribute of an Oracle session that you may set at different levels, depending upon the type of diagnostic output you wish to obtain. Many database administrators are already familiar with Oracle's `sql_trace` facility, which emits performance information about Oracle `parse`, `execute`, `fetch`, `commit`, and `rollback` database calls. Using `sql_trace` is actually the equivalent of using event 10046 set at level 1.

However, there are entire categories of system performance problems that cannot be diagnosed with level-1 data alone. For example, you cannot unambiguously diagnose and repair a system plagued by contention for latches, locks, networks, or even storage devices by looking only at SQL trace data. Event 10046 takes Oracle performance instrumentation a significant step further by detailing the Oracle kernel's executions of over two hundred internal function calls.

Upon this additional timing detail, Hotsos researchers have constructed an efficient method that reliably solves system performance problems in most situations with no further data collection required beyond the initial event 10046 data. Oracle insiders have used event 10046 data since the early 1990s to diagnose system performance problems. However, the broader community of database administrators have not embraced 10046 until recently, for at least the following reasons:

- *Technology.* The syntax required to activate event 10046 is cumbersome and not well documented. In the absence of good tools to help analyze the output, people have found event 10046 data difficult to interpret. Furthermore, until recently, insufficient research had been performed to verify whether performance optimization methods based upon event 10046 data could actually be reliable.[2]

- *Culture.* There is cultural intertia at work as well. Established experts have little external motive to overhaul their ways of working. Newcomers to the field of Oracle performance optimization are probably most influenced by the most popular "Oracle tuning" authors, who haven't improved the Oracle performance state of the art since the 1980s.

Oracle professionals are attacking both fronts today. Oracle Corporation is improving the accessibility of event 10046 data with enhancements to the *Oracle Trace* facility in

release 9. And through presentations and publications delivered internationally, thousands of Oracle database administrators are finally receiving the message that optimizing user response times is much better for business than optimizing system ratios.

The performance diagnostic value of 10046 data is extraordinary. This document is designed to help the typical database administrator employ this tool in the quest for faster, more efficient, and hence more scalable Oracle systems.

3. Levels of Event 10046 Tracing

You can think of the event 10046 "level" attribute associated with an Oracle session as a 4-bit flag whose bits have the following meanings:

Table 1. Event 10046 tracing levels

Level		Function
Decimal	**Binary**	
1	0001	Emit statistics for `parse`, `execute`, `fetch`, `commit`, and `rollback` database calls (standard `sql_trace`)
2	0010	Unknown
4	0100	Emit values for SQL bind variables (also called "placeholders")
8	1000	Emit statistics for Oracle kernel internal function calls (also called "wait events") listed in `v$event_name`

These levels can be combined as if by a bitwise `or` function to produce combinations of data in an Oracle trace file. For example, a level-12 trace combines the effects of level-4 and level-8 tracing. Strangely, activating any non-zero tracing level also activates level-1 tracing. Therefore, tracing at levels 4, 8, and 12 are exactly equivalent to tracing at levels 5, 9, and 13, respectively: all these levels include the standard `sql_trace` output. Performance analysts who use event 10046 tracing almost always use either level 8 (wait events) or level 12 (wait events plus bind values). Specifying level 9 or 13 provides the modest advantage of making your 10046-savvy colleagues think that you know something they don't know. We presently know of no other benefit to adding the "1".

Warning

Note that the level-4 tracing option opens new possibilities for breaches of data security, because it records database data values into text files in your OS filesystem.

4. Activating and Deactivating Event 10046 Tracing

In order for your event 10046 data to be valuable, you must first ensure that the session is capable of computing its own timing statistics. If you fail to enable timed statistics computations, then your performance data will contain zeros instead of true timing information. This of course makes it much more difficult to use the data to improve end-

user response times. You manipulate a session's ability to time itself with the Oracle parameter `timed_statistics`. This parameter has been session-modifiable since release 7.3.

Next, for the resulting output to be complete, you must ensure that the size of the session's trace file is not arbitrarily restricted. You manipulate a session's maximum trace file size with the Oracle parameter `max_dump_file_size`. This parameter has also been session-modifiable since release 7.3.

There are several ways to activate event 10046 for a chosen session. Which one to use is primarily a function of whose session you wish to trace. Tracing your own session is straightforward; simply use `alter session` commands to specify the event and level of tracing that you desire. For someone else's session (for example, an application user's session), Oracle provides packaged procedures to do the job.

Deactivating event 10046 is simple, requiring similar syntax to activation. However, in many cases the performance analyst should not explicitly deactivate 10046 data collection. The Oracle kernel writes certain data to the trace file only upon cursor close.[3] If you can allow a session to disconnect naturally, then there is no need to deactivate tracing for the session. If not, however, then you should deactivate tracing explicitly, such as in the case of a persistent service (an Oracle Applications concurrent manager process, for example).

4.1. Tracing Your Own Session

Tracing your own session is a simple process that is the same, regardless of whether your application is SQL*Plus, Pro*C, OCI, or any other client program. To activate and explicitly deactivate level-8 tracing for your own Oracle session, execute the following SQL statements:

```
alter session set timed_statistics=true
alter session set max_dump_file_size=unlimited
alter session set events '10046 trace name context forever, level 8'
...
alter session set events '10046 trace name context off'
```

Note that some Oracle ports (notably Oracle7 and Oracle8*i* for Microsoft Windows) do not support the `unlimited` keyword value. For these ports, simply set `max_dump_file_size` to a large integer. On the 32-bit implementations of Oracle in our laboratory, the maximum value you can specify is $2^{31} - 1 = 2{,}147{,}483{,}647$.

4.2. Tracing Someone Else's Session (Oracle release 8.1.6 and newer)

To trace someone else's session, first your must obtain the session id (`v$session.sid`) and serial number (`v$session.serial#`) for the session you wish to trace. Then activate tracing for the chosen session. To activate and explicitly deactivate level-8 tracing for the specific session identified by (*`sid`*, *`serial#`*), execute the following SQL statements:

```
exec sys.dbms_system.set_bool_param_in_session(sid, serial#,
'timed_statistics', true)
```

```
exec sys.dbms_system.set_int_param_in_session(sid, serial#,
'max_dump_file_size', 2147483647)
exec sys.dbms_system.set_ev(sid, serial#, 10046, 8, '')
...
exec sys.dbms_system.set_ev(sid, serial#, 10046, 0, '')
```

Some Oracle Metalink bulletins describe `dbms_system.set_ev` as "unsupported," describing that customers should use `dbms_support.start_trace_in_session` instead. However, `dbms_support` is particularly difficult to obtain, and we have learned that `dbms_support.start_trace_in_session` is actually implemented as a call to `set_ev` anyway.

By contrast, `dbms_system` is shipped with the standard Oracle distribution. Oracle support analysts recommend using `set_ev` in several Metalink bulletins, and Hotsos customers have used `set_ev` in hundreds of performance improvement projects without incident.

4.3. Tracing Someone Else's Session (Oracle releases prior to 8.1.6)

The packaged procedures to set Oracle parameters in someone else's session are not distributed with Oracle releases prior to 8.1.6. However, it is easy to set the necessary parameters instance-wide. Of course, you have the option of changing them again when you are finished tracing. However, it is *not* possible through Oracle9*i* to read the current value of another session's session-level parameter setting; it is thus impossible to record a parameter's current value so that it can be restored later to that value.

To activate and explicitly deactivate level-8 tracing for the specific session identified by (*sid*, *serial#*), execute the following SQL statements:

```
alter system set timed_statistics=true
alter system set max_dump_file_size=unlimited
exec sys.dbms_system.set_ev(sid, serial#, 10046, 8, '')
...
exec sys.dbms_system.set_ev(sid, serial#, 10046, 0, '')
```

5. Finding Your Trace File

Once you trace a session, your next task is to identify the trace file (or files) where your trace data were written. The value of the Oracle parameter *user_dump_dest* contains the name of the directory into which *user* sessions write their trace files (*user* sessions are sessions for which `v$session.type='USER'`). The value of the Oracle parameter `background_dump_dest` contains the name of the directory into which *background* sessions write their trace files (*background* sessions are sessions for which `v$session.type='BACKGROUND'`).

Once you have identified the directory in which your trace file resides, you need to identify the name of the file. Unfortunately, the various porting groups at Oracle Corporation have not settled upon a file naming standard. Therefore, it can be difficult to write a platform-independent tool that can predict what the trace file for a given session will be called. For example, we have observed the following:

Table 2. Apparent Oracle trace file naming conventions

Oracle port	Trace file name pattern
Linux	`ora_`*`SPID`*`.trc`
HP-UX	`ora_`*`SPID`*`_`*`SID`*`.trc`
Compaq OpenVMS	`ora_`*`INSTANCE`*`_(fg|bg)_oracle_`*`SID`*`.trc`
Microsoft Windows	`ora_`*`NUMBER`*`.trc`

The most reliable way to determine the name of your trace file is as follows:

Procedure 1. Procedure for finding the trace file for a given OS process id (`spid`)

1. Identify the directory in which your trace file was written.
2. List the contents of that directory, ordered by descending file modification time (for example, using **ls -lt** in Unix).
3. For each file in that list,
    a. Search the preamble of the file for the line containing the string "`pid`" (Unix, Linux, OpenVMS) or "`thread id`" (MSWin32). The preamble consists of all the lines up to the line that begins with the string "`***`".
    b. If the number following this string matches the value of `v$process.spid` for your chosen session, then you have found the correct trace file; stop searching.
4. If you have exhausted the file list without finding a matching `spid`, then the file you are looking for does not exist.

The Hotsos *Trace File Dæmon* (`trcfiled`), part of the free Sparky distribution, is open-source Perl code that executes exactly this function. This algorithm is very efficient even for directories with large numbers of trace files in them, as long as the trace file you are seeking actually exists.

6. Interpreting Your Trace File

Once you have identified your trace file, this is where the real progress begins. For most cases, the trace file contains all of the information you will need to diagnose and repair the session's performance problem. However, it can be difficult to interpret raw trace data, especially when the size of the raw trace file is extremely large. There are several ways that you can interpret your trace file:

- Inspect the file visually. There is excellent documentation on Oracle Metalink[4] that describes the meaning of each field in the raw trace data. However, it is difficult to interpret raw trace data by inspection when the raw trace file is extremely large.

- Construct a software tool using, for example, Perl or awk, that summarizes the content of the file for you. The Hotsos Clinic includes instruction that helps facilitate this process.

- Use a prepackaged software tool to summarize and format the raw trace data. A few such tools are available today, including Oracle's `tkprof` tool that is shipped as part of the Oracle standard distribution. However, `tkprof` ignores level-4 and level-8 trace data until release 9.0.1. Even in 9.0.1, `tkprof` still produces wait event information in a very ratio-centric way. A Turkish company named Unal Bilism provides online access to the itrprof SQL analyzer, which performs a similar function. The Hotsos Profiler is the tool that Hotsos developers have constructed to maximize the value of event 10046 data.

7. Using Event 10046 with Multiplexing Architectures

Oracle's session-based model for SQL tracing is not particularly well-suited for the tracing of transactions that are multiplexed across Oracle server processes, either by the Oracle Multi-Threaded Server (MTS) or application server middle tiers. The 10046 data are just as valuable in this architecture as in any other; the issue is that it becomes more difficult to obtain.

Fortunately, this appears to be an area in which Oracle is paying particular attention. The Oracle Trace documentation for Oracle release 9.0.1 is definitely encouraging with its description of the *migration* Oracle Server Event.[5]

7.1. Oracle Multi-Threaded Server (MTS)

Oracle Corporation introduced its Oracle Multi-Threaded Server (MTS) in release 7.0 to reduce the number of OS process instantiations required to service large numbers of connected users. Instead of every Oracle connection request motivating a `fork` and `exec` of a new dedicated Oracle server process, Oracle MTS allows a large number of user processes to share the services of a smaller number of server processes.

The challenge this presents to 10046 data users is that the MTS architecture causes the database calls issued by a single user to be distributed across potentially several different Oracle trace files. For example, a client program's `execute` call may be serviced by shared server process number 3, and hence the `execute` call's 10046 data will be listed in trace file number 3. The same client program's first `fetch` call might be serviced by shared server number 7, and hence the `parse` call's 10046 data will be listed in trace file number 7. The program's next `fetch` call might be serviced by a different shared server process again, and so on.

As a result, to assemble all of the 10046 data for a given Oracle user connection in an MTS architecture requires an operation that very closely resembles an Oracle sort-merge join operation executed upon multiple trace files. Each session's activity in an Oracle trace file is identified in the trace file text by the motivating Oracle connection's unique session id. Therefore, it is straightforward to create a “filter” that returns only those parts of a trace file that are associated with a specified session. Furthermore, each session's

segment of 10046 data in a trace file has a timestamp marking when the segment was written to the trace file. Therefore, it is also straightforward to sort a session's 10046 data obtained from several trace files into a time-ordered sequence of events.

7.2. Application Server Middle Tiers

Connection pooling architectures can pose a difficulty to the collection of 10046 trace for a specified user session by concealing the origin of a database call from the database. Figure 1 shows how. In this figure, the application server middle tier accepts HTTP requests from the five browsers in the user community. Many of these HTTP requests motivate the application server to issue database `parse`, `execute`, and `fetch` calls. However, the application server conceals the identity of the user from the database. Therefore, unless the application source code or the application server is specifically instrumented to record the identity of a 10046 trace file line, the reader of the 10046 data will have no way of determining which user motivated which lines.

Figure 1. A connection pooling application architecture

In Figure 1, different database calls motivated by the browser at IP address 150.121.1.104 have been processed by each of the three different Oracle server processes, resulting in data corresponding to browser 104 transactions (marked red) in each of the three trace files. However, there is no way that the reader of each trace file can determine which browser has motivated the database calls represented by the 10046 data. The application server has concealed the identity of browser 104 from the database.

There are several ways to use 10046 data in spite of the difficulty imposed by connection pooling:

- *Instrument the application.* The Oracle Trace facility provides the means for tagging a user transaction so that it can be identified within the database performance data. The downside of this option is that instrumenting your application can be burdensome, if not practically impossible (e.g., in the case of purchased, packaged applications). The upside is that according to the Oracle9*i* documentation, Oracle Trace finally makes available the types of information that previously only event 10046 revealed.

- *Aggregate across application users.* One of the distinctions of Hotsos training is our admonition that averaging performance statistics across heterogeneous transaction types adds ambiguity and hence difficulty to a performance problem repair project. However, for *homogeneous* transaction types, averaging can be a powerful tool. If all five of the browsers depicted in Figure 1 are executing the same type of transaction, then an analyst can obtain perfectly legitimate performance data by examining the content of any or all of the the trace files shown.

  For example, if all of the browsers had been running the same order entry form, entering the same types of orders for some 10-minute interval, then aggregating

all of the trace files' content for that 10-minute interval and dividing by the number of orders will produce legitimate *per-order* performance statistics. Likewise, in this type of heterogeneous environment, examining the trace data for any single trace file should reveal performance characterstics consistent with the average.

Figure 2. Aggregating across similar transactions in a connection pooling environment

- *Isolate the application.* Many times, it is possible to identify an end user's trace data by isolating that user's database activity to a single dedicated Oracle server process. For example, if you can create the scenario depicted in Figure 3, then you have solved the data collection problem for browser 104.

  The downside of this method is that if the performance of the original application server program was in fact the root cause of your performance problem, then you will have obscured that issue by routing your transactions through a different middle tier process. However, if performance is drastically better for browser 104 after isolating it onto its own application server process, then you have observed a strong indication that the performance problem was in the middle tier (and not in the Oracle database) to begin with.

  Figure 3. Isolating a user in a connection pooling environment

8. Known 10046-Related Oracle Side Effects

Using Oracle's 10046 trace data has proven to be remarkably free of side effects The only negative side effects that we have found to date are listed here:

8.1. Oracle bug 1210242

This bug affects many Oracle8*i* releases. It causes Oracle not to share SQL statements properly in the library cache. Patches are available. Consult Oracle Metalink for details.

8.2. Rumors of Oracle instance failure

Tracing the Oracle PMON background process is reputed to cause ORA-600 errors and possibly instance failures. However, we have not yet found a Metalink bug number that specifically links instance crashes with event 10046. The good news is that the rumors of trouble are moot in most situations; the need to execute a 10046 trace upon PMON should be exceedingly infrequent.

9. Examples

Figure 4 shows a complete Oracle trace file that includes event 10046 data. Lines that begin with the string "WAIT" are specifically generated by the level-8 attribute of the event 10046 trace. Other lines are present even with a standard sql_trace=true setting for the session.

You can see from the trace data that the session motivating these event 10046 data lines was a SQL*Plus session that executed the following statements:

```
alter session set events '10046 trace name context forever, level 8'
select count(*) from a
alter session set events '10046 trace name context off'
```

The extraordinary value of Oracle event 10046 output is how clearly it illustrates exactly what the Oracle kernel is doing on behalf of your database calls. There is simply no better way to understand exactly what an Oracle session is doing with your users' time.

Figure 4. A complete 10046 level-8 trace file for a simple session

```
Dump file C:\oracle\admin\ora817\udump\ORA00684.TRC
Tue Feb 12 15:22:03 2002
ORACLE V8.1.7.0.0 - Production vsnsta=0
vsnsql=e vsnxtr=3
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Instance name: ora817

Redo thread mounted by this instance: 1

Oracle process number: 9

Windows thread id: 684, image: ORACLE.EXE


*** 2002-02-12 15:22:03.426
*** SESSION ID:(8.1565) 2002-02-12 15:22:03.416
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====================
PARSING IN CURSOR #1 len=69 dep=0 uid=20 oct=42 lid=20 tim=40950208
hv=589283212 ad='38d4734'
alter session set events '10046 trace name context forever, level 8'
END OF STMT
EXEC #1:c=1,e=1,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40950208
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 444 p1=1111838976 p2=1
p3=0
=====================
PARSING IN CURSOR #1 len=23 dep=0 uid=20 oct=3 lid=20 tim=40950654
hv=2180270315 ad='38d7678'
select count(*) from a
END OF STMT
PARSE #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40950654
EXEC #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40950654
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
```

```
WAIT #1: nam='file open' ela= 0 p1=0 p2=0 p3=0
WAIT #1: nam='db file scattered read' ela= 6 p1=7 p2=10 p3=7
WAIT #1: nam='db file scattered read' ela= 4 p1=7 p2=17 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=25 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=33 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=41 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=49 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=57 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=65 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=73 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=81 p3=8
WAIT #1: nam='db file scattered read' ela= 4 p1=7 p2=89 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=97 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=105 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=113 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=121 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=129 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=137 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=145 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=153 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=161 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=169 p3=8
WAIT #1: nam='db file scattered read' ela= 3 p1=7 p2=177 p3=8
WAIT #1: nam='db file scattered read' ela= 3 p1=7 p2=185 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=193 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=201 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=209 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=217 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=225 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=233 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=241 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=249 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=257 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=265 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=273 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=281 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=289 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=297 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=305 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=313 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=321 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=329 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=337 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=345 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=353 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=361 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=369 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=377 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=385 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=393 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=401 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=409 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=417 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=425 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=433 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=441 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=449 p3=8
```

```
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=457 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=465 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=473 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=481 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=489 p3=8
WAIT #1: nam='db file scattered read' ela= 1 p1=7 p2=497 p3=8
WAIT #1: nam='db file scattered read' ela= 0 p1=7 p2=505 p3=3
FETCH #1:c=10,e=52,p=498,cr=498,cu=5,mis=0,r=1,dep=0,og=4,tim=40950706
WAIT #1: nam='SQL*Net message from client' ela= 0 p1=1111838976 p2=1
p3=0
FETCH #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=40950706
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 961 p1=1111838976 p2=1
p3=0
STAT #1 id=1 cnt=1 pid=0 pos=0 obj=0 op='SORT AGGREGATE '
STAT #1 id=2 cnt=262144 pid=1 pos=1 obj=3176 op='TABLE ACCESS FULL A '
=====================
PARSING IN CURSOR #1 len=56 dep=0 uid=20 oct=42 lid=20 tim=40951667
hv=3475487367 ad='38d2dec'
alter session set events '10046 trace name context off'
END OF STMT
PARSE #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40951667
EXEC #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40951667
```

10. Resources

For detailed information about how to interpret raw trace data, visit Oracle Metalink bulletin 39817.1.

Learn about the Hotsos Profiler at www.hotsos.com/products/profiler.

Learn about Sparky, the free event 10046 data collector, by visiting www.hotsos.com/products/sparky.

For information about Hotsos Clinic, a three-day training event dedicated to instruction on the Hotsos method for rapid performance problem diagnosis and resolution, visit www.hotsos.com/training/clinic.

11. Acknowledgments

Our special thanks to Juan Loaiza, Roderick Mañalac, Anjo Kolk, Virag Saksena, and Mogens Nørgaard. These gentlemen are prominent amoung the pioneers who invented, implemented, documented, and inspired people to use Oracle's event 10046 data.

---

[1] You can find a listing of all such pseudo-error debugging events for your Oracle environment by viewing the file on your system called `$ORACLE_HOME/rdbms/mesg/oraus.msg`. Unfortunately, Oracle Corporation appears not to provide this file on its Microsoft Windows ports.

[2] The Hotsos Clinic three-day training event describes in technical detail why event 10046 output is far more valuable than most people once believed.

[3] Specifically, the Oracle kernel writes `STAT` lines, which provide SQL execution plan information, only when a cursor closes.

[4] "Interpreting raw sql_trace and dbms_support.start_trace output," 1998. Oracle Metalink document id 39817.1.

[5] See the Oracle9*i* documentation at Oracle Technet for more information.