Oracle Trace: The Swiss Army Knife of Diagnostic Tools

by Arup Nanda

Introduction

One of the hidden jewels in Oracle's offering is the Oracle Trace, a rarely used tool that can be like a Swiss Army Knife to a DBA for performance tuning and evaluation. There are several reasons why Oracle Trace is seldom used, primarily the lack of a simple user interface and the inadequate documentation. Although the purpose of this article is not to fill the gap in documentation, as it takes much beyond the usual few pages to describe the power of this tool, I will try to provide as much information as possible on how it can be used. It is very important to distinguish Oracle Trace from the more common SQL Trace, which offers a similar but more limited functionality in certain areas.

The concept of Oracle Trace can be best described through an example: Assume you are a DBA of a large Oracle shop and you have been asked to diagnose a performance-related problem in an application that runs slower than usual. The application is canned so you would not be able to place an sql trace in there directly. There are a few tricks you can pull to make a diagnosis: you could take a STATSPACK snapshot before and after the run and a SQL_TRACE using DBMS_SYSTEM, DBMS_SUPPORT, or *oradebug*. STATSPACK will perhaps tell you that the database waited on several wait events, among which are notably the *db file scattered read*, *db file sequential read*, *buffer busy waits* and *free buffer waits* that constituted about 90 percent of the waits. SQL_TRACE will generate a trace file that can be run through TKPROF; which will generate a formatted trace file, which is perhaps great for diagnosing optimizer plans but not wait events.

Since the waits are due to excessive buffer manipulation of a segment, further diagnosis must unearth the segment in question. However, the sql_trace and tkprof combination does not provide details at that level. One way to get that is by setting an event 10046 making the raw trace file contain the waits information. Although it does contain the waits data, tkprof ignores it while processing the trace file. The result is that the raw trace file contents are not compiled into a readable format making the analysis difficult and somewhat impossible.

Oracle Trace just makes this task trivial by both collecting and analyzing. First, we will start Oracle Trace and let the database run. We define the scope of the collection – user level, database level and the exact nature of the collections. After a while, we will shut off the trace collection. Unlike sql trace, it does not produce a trace file per session, but a somewhat unreadable file for the entire collection. We will then run the generated trace file through a formatter, which loads the data into Oracle database tables. After that, there are several ways the data could be analyzed – using the simple sql to the *Oracle Enterprise Manager Trace Data Viewer*. In the same example, we would use a query like the following:

```
SELECT DISTINCT tablespace_name
FROM DBA_DATA_FILES F, WAITS W
WHERE W.DESCRIPTION = 'db file scattered
read'
AND F.FILE_ID = W.P1
```

This will instantly tell us the tablespace where this wait event occurred. Since this is exactly same as the v$session_wait view's P1, P2 and P3 parameters, you could even drill down to find out the exact segments where the problem occurred.

I am sure you now appreciate the value of Oracle Trace tool over several other tools in offering the same information. Let's go about setting this up and master the tool as we go along. In this document, ORACLE_HOME is referred to as OH for brevity.

Facilities

Each type of event that occurs in an Oracle database is called a *facility* in the OT parlance. There are two different kinds of events: *Point Events*, where the event occurs as a point in time, e.g., *Connection Established to Server*; and *Duration Events*, where the event starts and ends with something, e.g., *Execution of a SQL Statement*. Oracle Trace can intercept and report 13 events, 17 in Oracle9i. A few examples of events are information about waits, sql statements and cache IO details. A full listing of events can be obtained from the manual. Each facility is controlled through Oracle-supplied files called *Facility Definition Files* (FDF), which are located in `$OH/otrace/admin/fdf` directory.

Init.ora Setting

The following parameters in init.ora control the collection settings.

oracle_trace_enable	It should be set to TRUE; can also be set via ALTER SESSION.
oracle_trace_collection_name	It should either be left unset, in which case it defaults to “oracle”, or be

	specified as "" (a blank string). The exact name for a collection will be set in the configuration file.
oracle_trace_collection_path	It sets the directory where the files that store the trace data are located. This defaults to $OH/otrace/admin/cdf
oracle_trace_facility_name	The facility for which the data is collected, defaulting to oracled, meaning Oracle Default Events.
oracle_trace_facility_path	Defaults to $OH/otrace/admin/fdf, which contains the FDF files of the facilities.

Database User

This user stores the trace collection tables. Create a user called TRACESVR (or any other name) with a password TRACE and a separate default tablespace, called, say, TRACEDATA. Grant RESOURCE role to this user.

Setting up the Collection

The collection is controlled by a program called `otrccol` in `$OH/bin` directory. This program takes a configuration file as a parameter. This file can be kept in any directory, but for administrative convenience I generally keep them in `$OH/otrace/admin` directory. Here are the parameters and their descriptions.

col_name	The collection name. It can be anything meaningful.
cdf_file	The name of the collection definition file. It specifies what is to be collected in that collection. It should be given the same name as the collection with a cdf extension.
dat_file	The name of the file where the output of the collection is kept. Generally, it's given the same name as collection with dat extension.
fdf_file	The facility definition file to use. Use any of the FDF files from the $OH/otrace/fdf directory, or the directory defined by the init.ora parameter.
max_cdf	The size in bytes of the trace file that will be generated. A value of 0 means no limit; a positive value up to 2 GB indicates when the size is reached the collection should stop. If this is a negative number, the file will wrap around deleting the oldest information.
buffer_size	Oracle keeps data up to this value in bytes before it writes to the trace data file.
regid	This is an important line and is described below in detail.
username	The user name that will be used for formatting. This can also be set at the command.
password	Password of the above user.
service	Database Service Name
full_format	While formatting, the formatter can use the entire data contained in data files or only new data elements, which is controlled by this parameter. Set this to 1 for entire data.

Registration

The most important parameter is *regid* in the configuration file that controls what to collect. It contains a line in the format

```
1 192216243 <cross facility number> <cross facility value> 5 <Database SID>
```

where <*cross facility number*> is the cross facility we are interested in. If we are interested in server wait events, we will use a number 7 here. The <*cross facility value*> is the database event code we will trap. The event code corresponds to the EVENT# column of the V$EVENT_NAME view. For example if we are tracing *db file scattered read* event, we will get the *event#* first.

```
SELECT EVENT# FROM V$EVENT_NAME WHERE NAME =
'db file scattered read';
```

In our case, this is 94; so we will use 94 as the <*cross facility value*>. This parameter can also be repeated for each wait event. If we are interested in also tracing *db file sequential read, buffer busy waits and free buffer waits events* we have to get their event numbers, which turn out to be 95, 68 and 75 respectively. Finally, our configuration file `trace.cfg` looks like this:

```
col_name= ananda
cdf_file= ananda.cdf
dat_file= ananda.dat
fdf_file= waits.fdf
max_cdf= -10485760
buffer_size = 1048576
regid= 1 192216243 7 95 5 cgr1
regid= 1 192216243 7 94 5 cgr1
regid= 1 192216243 7 68 5 cgr1
regid= 1 192216243 7 75 5 cgr1
username= tracesvr
password= trace
service= AND9
full_format= 1
```

Starting the Collection

Now that the setup is over, we will start the collection. The following line given from a command prompt will start the collection:

```
otrccol start 99 trace.cfg
```

The number 99 is the jobid, used for no particular reason. You can specify any positive non-zero number there. After a while, stop the collection using

```
otrccol stop 99 trace.cfg
```

If you look at the collection files directory (as defined in the init.ora, usually `$OH/otrace/admin/cdf`) you will notice two huge files have been generated, named `ananda.cdf` and `ananda.dat`. These contain the data for the trace. To check whether the collection is going well, you could issue

```
otrccol check ananda.cdf
```

You can use the following command to delete the collection when they are formatted and analyzed:

```
otrccol dcf ananda ananda.cdf
```

Formatting

Now comes the time to view the data. The data from the trace files can be formatted to a report using this command:

```
otrcrep –w132 ananda.cdf
```

This produces several text files that are 132 columns wide, providing much of the output in a report format for quick viewing; however, the report is not particularly useful. The better option is to load the data into database tables. This can be achieved by the command

```
otrccol format trace.cfg
```

This creates several tables in TRACESVR schema and loads data from the trace files. These tables are named in a somewhat user-unfriendly way, e.g., `V_192216243_F_5_E_13_8_1`. The understandable synonyms are created by running under TRACESVR the script `$OH/otrace/demo/otrsyn.sql`.

Viewing the Trace Data

The easiest way to see the data is using sql. The synonym WAITS in TRACESVR schema contains the data about the wait events the session experienced. A simple query like the following will get us the information:

```
SELECT p1 "File ID", p2 "Block ID",
      p3 "Reason"
FROM WAITS WHERE description = 'db file
scattered read';
```

We can then join the file id and block id to the DBA_EXTENTS to find the segment name.

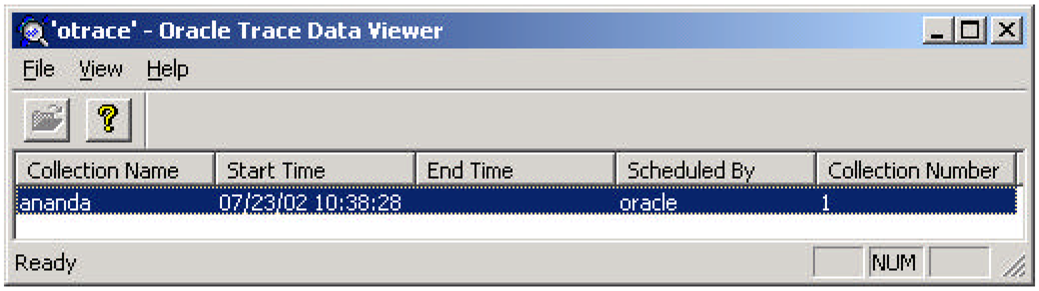
```
SELECT SEGMENT_NAME FROM DBA_EXTENTS WHERE
FILE_ID = <file_id> and <block_id> BETWEEN
BLOCK_ID and BLOCK_ID + BLOCKS;
```

This will get us all the information we wanted to know about the segments that contributed to the problem. The other columns in the WAITS synonym are also useful. SESSION_INDEX and SESSION_SERIAL denote the SID and SERIAL# from V$SESSION for the session that was affected. TIMESTAMP provides a clue as to when exactly it happened, which can then be cross checked with the performance problems times; or the data can be grouped on TIMESTAMP to know what time most of the problems appeared. The columns starting with the word CROSS_FAC are cross facility numbers not used in our analysis here.

The other synonyms that were created also throw light into what happened during the process. The synonym CONNECTS records all connections, the OS User, the machine and much more information to link with the WAITS table to get a meaningful profile of the application that was just run. The DISCONNECTS table provides information on when the session was disconnected. It also records the time in nanoseconds (TIMESTAMP_NANO column), making the duration calculation extremely fine-grained.

Trace Data Viewer

The other option to see the trace data is through the Trace Data Viewer (TDV) in Enterprise Manager. This provides a graphical view of the data and click-to-drilldown facility to analyze data. You can start TDV from *OEM's Menu Bar -> Tools -> Diagnostic Pack* submenu. After connecting, you will see the following screen (Figure 1).



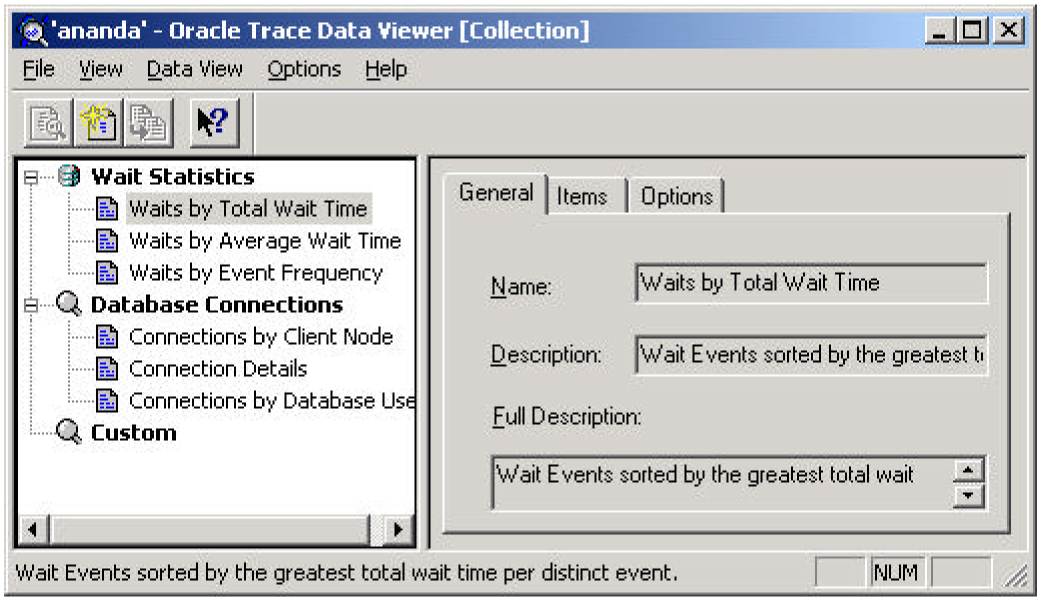
Collection Name	Start Time	End Time	Scheduled By	Collection Number
ananda	07/23/02 10:38:28		oracle	1

Clicking on the collection brings up the data in that collection as shown in the screen below (Figure 2).

Clicking on *Waits by Total Wait Time* gives us the view (Figure 3).

Drilling down further on the *db file scattered read*, we could see the details we were looking for like in Figure 4.

We saw the same figures we saw in the sql approach but in a graphical manner. You can use either one, based on your comfort level.



'ananda' - Oracle Trace Data Viewer [Collection]

File View Data View Options Help

- Wait Statistics
  - Waits by Total Wait Time
  - Waits by Average Wait Time
  - Waits by Event Frequency
- Database Connections
  - Connections by Client Node
  - Connection Details
  - Connections by Database Use
- Custom

General | Items | Options

Name: Waits by Total Wait Time

Description: Wait Events sorted by the greatest t

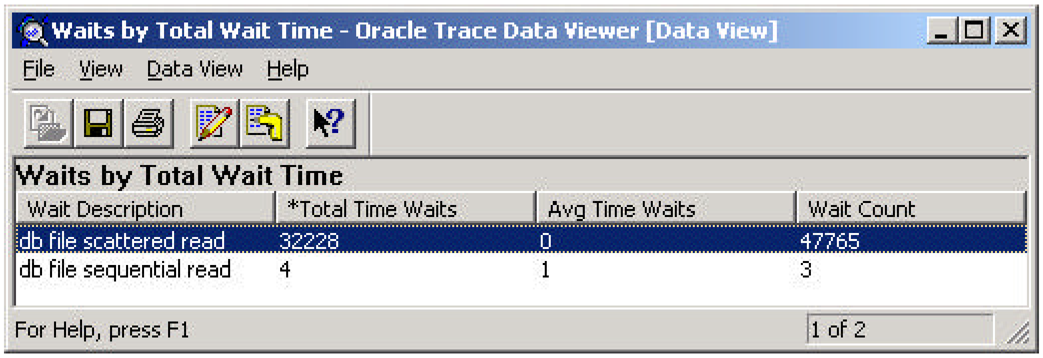
Full Description:

Wait Events sorted by the greatest total wait

Wait Events sorted by the greatest total wait time per distinct event. NUM

Other Uses

So far, we have seen the events where the sessions waited on



Waits by Total Wait Time - Oracle Trace Data Viewer [Data View]

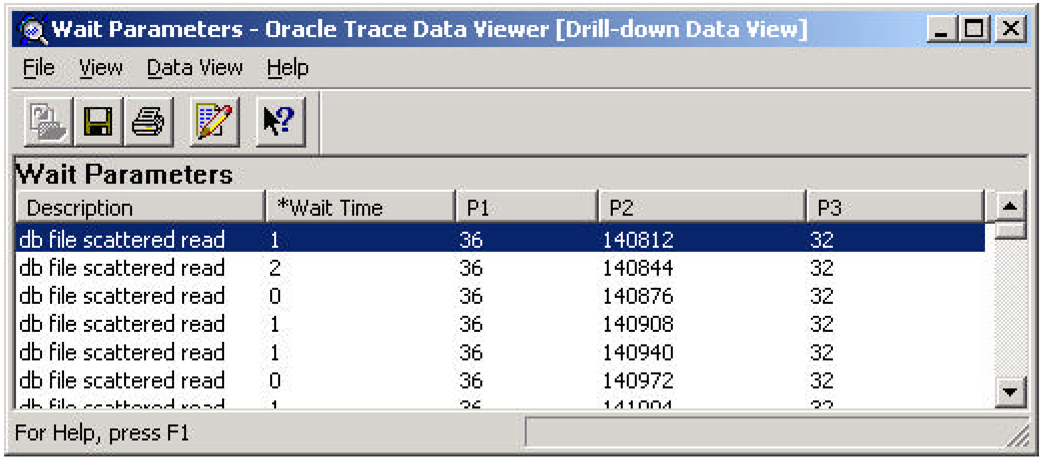
File View Data View Help

Waits by Total Wait Time

Wait Description	*Total Time Waits	Avg Time Waits	Wait Count
db file scattered read	32228	0	47765
db file sequential read	4	1	3

For Help, press F1 1 of 2

some database event. There are several more things you can do using Oracle Trace in many other instances. If you want to get the SQL statements that were executed in the sessions, you could use `sql_txn.fdf` as the value of the `fdf_file` parameter in the configuration file. This will produce all the SQL Statements, their plans and statistics, much like what you would get from sql_trace and tkprof, but loaded into database tables for easier analysis and graphical viewing.



Wait Parameters - Oracle Trace Data Viewer [Drill-down Data View]

File View Data View Help

Wait Parameters

Description	*Wait Time	P1	P2	P3
db file scattered read	1	36	140812	32
db file scattered read	2	36	140844	32
db file scattered read	0	36	140876	32
db file scattered read	1	36	140908	32
db file scattered read	1	36	140940	32
db file scattered read	0	36	140972	32
db file scattered read	1	36	141004	32

For Help, press F1

Another very useful application of Oracle Trace is tracing only for certain users. For example, if you want to trace the sessions of only user ANANDA in the database, you can find out the USER_ID of the user from database by issuing

```
SELECT USER_ID FROM
DBA_USERS WHERE USERNAME =
'ANANDA';
```

Say this returns 12. You can use the regid parameter in the configuration file as

```
regid= 1 192216243 6 12 5
AND9
```

The third parameter denotes the *cross facility number* and a value of 6 indicates the database user ids. The fourth parameter is the user id, which is 12 for the user ANANDA.

Conclusion

Oracle Trace provides an extremely powerful and versatile tool for several types of diagnoses in the Oracle database. The possibilities of what can be done are plenty and it just needs to be exploited to bring this to reality. This article did not attempt to cover all the facilities offered by the tool, but rather to introduce it to you with the hope of steering you towards using it to achieve tangible goals.

About the Author

Arup Nanda has been an Oracle DBA for more than 9 years specializing in Performance Tuning, Design and Datawarehousing. He is the founder of Proligence (www.proligence.com), a New York area based company providing specialized Oracle DBA services like Replication Setup, Parallel Server and Real Application Server Installs and Datawarehouse Design and Development, among others. When not pulling his hair to resolve an Oracle problem or presenting at a User Group, Arup loves playing tennis and oil painting. He will appreciate feedback on this article at arup@proligence.com.