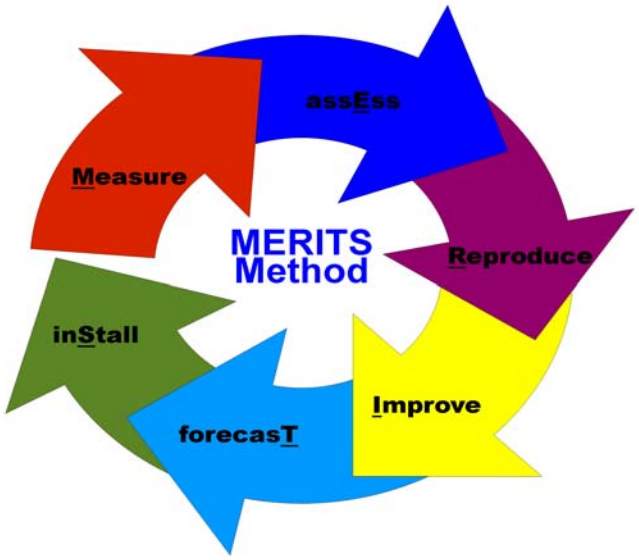
AS-SYSTEME

MERITS Profiler User Guide

Release 1.2.1

MERITS Profiler User Guide



Author: Norbert Debes, AS-SYSTEME GmbH
MERITS Profiler Version 1.2.1, March 2012

Licensing

The MERITS Profiler ships with an Oracle release 11.1.0.7 JDBC driver. Several components from the Apache Software Foundation are also used by the Profiler. By using the MERITS Profiler, you agree to be bound to the terms set forth by the Oracle OTN License Agreement as well as the Apache License. Both licenses are available in <MPROF_HOME>/contrib, where MPROF_HOME is the Profiler installation directory. You may not use the MERITS Profiler if you don't agree to one or both of the license agreements.

Software Requirements

The Profiler is built with the latest Java technologies. It runs on any platform that supports Java and requires the following software:

- Java 1.6 Standard Edition (SE) or newer run-time environment (JRE)

This Java release is available for download as JRE 6 at http://java.sun.com/javase/downloads/index.jsp. Use the command java -version to find out what version of Java is installed on a system:

```
C:\> java -version
java version "1.6.0_12"
Java(TM) SE Runtime Environment (build 1.6.0_12-b04)
Java HotSpot(TM) Client VM (build 11.2-b01, mixed mode, sharing)
```

Any version matching the one above or higher is expected to work with the Profiler.

Supported Oracle DBMS Releases

The MERITS Profiler supports the following Oracle DBMS releases:

Oracle10g Release 1 (10.1)

Oracle10g Release 2 (10.2)

Oracle11g Release 1 (11.1)

Oracle11g Release 2 (11.2)

There are no plans to support Oracle9*i* and earlier releases. An older profiler called ESQLTRCPROF supports Oracle9*i*. It ships with the book “Secret Oracle” (ISBN 978-1-4357-0551-7).

Instrumentation and Accounting by Module and Action

The MERITS Profiler supports accounting of response times and wait events per module and action. Module and action are identifiers that may be used to inform the Oracle DBMS engine about tasks that a database client executes. Module and action names appear in V$SESSION. If SQL trace is enabled they are also recorded in SQL trace files. The subsequent section on software instrumentation provides background information on setting module and action names. A third identifier, the so-called client identifier is also discussed. The client-identifier is ideally suited for performance diagnosis and troubleshooting of database applications that utilize connection pooling.

Software Instrumentation

The term *software instrumentation* refers to a programming technique, whereby a program is capable of producing an account of its own execution time. The ORACLE DBMS is heavily instrumented (wait events, timers, counters), however this instrumentation may be leveraged to a greater degree when a database client informs the DBMS of the tasks (module and action) it is performing. This section discusses trace file entries that are related to application instrumentation. The minimum SQL trace level for enabling entries discussed in this section is 1 (see Profiler parameter sql_trace_level on page 28).

Application Instrumentation Entries in Oracle10*g* and Oracle11*g*

Table 1 lists the instrumentation entries of Oracle10*g* and Oracle11*g* in alphabetical order along with the PL/SQL and OCI interfaces to generate them. Note that Oracle JDBC drivers have Java instrumentation interfaces which are more efficient than calling PL/SQL from Java. At the lowest level, application instrumentation is achieved with the Oracle Call Interface (OCI) function `OCIAttrSet` (see *Oracle Call Interface Programmer's Guide*).

Table 1: PL/SQL and OCI Interfaces for Instrumentation Entries

Trace File Entry	*PL/SQL Interface*	*OCIAttrSet Attribute*
`ACTION NAME`	`DBMS_APPLICATION_INFO.SET_MODULE`, `DBMS_APPLICATION_INFO.SET_ACTION`	`OCI_ATTR_ACTION`
`CLIENT ID`	`DBMS_SESSION.SET_IDENTIFIER`	`OCI_ATTR_CLIENT_IDENTIFIER`[a]
`MODULE NAME`	`DBMS_APPLICATION_INFO.SET_MODULE`	`OCI_ATTR_MODULE`

a. `DBMS_APPLICATION_INFO.SET_CLIENT_INFO` and the OCI attribute `OCI_ATTR_CLIENT_INFO` set `V$SESSION.CLIENT_INFO`. This setting is not emitted to trace files and cannot be used in conjunction with the package `DBMS_MONITOR`.

When running code such as the following in SQL*Plus, all three types of instrumentation entries are written to a trace file.

```
C:> sqlplus ndebes/secret@ten_g.oradbpro.com
Connected.
SQL> BEGIN
   dbms_application_info.set_module('mod', 'act');
   dbms_session.set_identifier(sys_context('userenv','os_user') ||
   '@' || sys_context('userenv','host') || ' (' ||
   sys_context('userenv','ip_address') || ')' );
END;
/
PL/SQL procedure successfully completed.
SQL> ALTER SESSION SET sql_trace=TRUE;
Session altered.
```

The resulting trace file contains lines such as these:

```
*** ACTION NAME:(act) 2007-08-31 18:02:26.578
*** MODULE NAME:(mod) 2007-08-31 18:02:26.578
*** SERVICE NAME:(orcl.oradbpro.com) 2007-08-31 18:02:26.578
*** CLIENT ID:(DBSERVER\ndebes@WORKGROUP\DBSERVER (192.168.10.1)) 2007-08-31 18:02:26.578
*** SESSION ID:(149.21) 2007-08-31 18:02:26.578
```

The value `orcl.oradbpro.com` of `SERVICE NAME` stems from the use of this string as the `SERVICE_NAME` in the definition of the Net service name `ten_g.oradbpro.com`.

These are the kinds of trace file entries that the Oracle10*g* TRCSESS utility searches for when used to extract relevant sections from one or more trace files. The sections that follow provide additional detail on the individual entries.

Module Name

The module name is intended to convey the name of an application or larger module to the DBMS. The default setting is `NULL`. SQL*Plus and Perl DBI automatically set a module name. The example below is from a SQL*Plus session:

```
*** MODULE NAME:(SQL*Plus) 2007-02-06 15:53:20.844
```

Action Name

An action name represents a smaller unit of code or a subroutine. A module might call several subroutines, where each subroutine sets a different action name. The default setting `NULL` results in a zero length action name:

```
*** ACTION NAME:() 2007-02-06 15:53:20.844
```

Client Identifier

Performance problems or hanging issues in three tier environments, where the application uses a database connection pool maintained by an intermediate application server layer, can be extremely cumbersome to track down. Due to the connection pool in the middle tier, performance analysts looking at V$ views or extended SQL trace files cannot form an association between an application user reporting a slow database and the database session or server process within the ORACLE instance serving a particular user. There is no way to find out which SQL statements are run on behalf of the complaining end user. Unless the application is properly instrumented, which is something that has eluded me in my career as a DBA.

The client identifier is the answer to this dilemma. The package `DBMS_SESSION` provides a means for an application to communicate an identifier that uniquely designates an application user to the DBMS. This identifier becomes the value of the column `V$SESSION.CLIENT_IDENTIFIER`. If SQL trace is enabled, this same identifier is also embedded in the SQL trace file. The format is:

```
*** CLIENT ID:(client_identifier) YYYY-MM-DD HH24:MI:SS.FF3
```

Client_identifier is the client identifier set by calling the procedure `DBMS_SESSION.SET_IDENTIFIER` from the application code. To extract trace information for a certain client identifier from one or more SQL trace files, `trcsess clientid=`*client_identifier* can be used. The line below shows an actual entry from a trace file. The client identifier used was ND. The entry was written on February 6th, 2007.

```
*** CLIENT ID:(ND) 2007-02-06 15:53:20.844
```

The maximum length of a client identifier is 64 bytes. Strings exceeding this length are silently truncated. When instrumenting applications with `DBMS_SESSION`, consider that the procedure `DBMS_SESSION.CLEAR_IDENTIFIER` does not write a `CLIENT ID` entry into the trace file, leaving the client identifier in effect until it is changed with `DBMS_SESSION.SET_IDENTIFIER`. When connection pooling is used, this may result in trace files where sections pertaining to different client identifiers are not delineated. The solution consists of setting an empty client identifier by passing `NULL` to the packaged procedure `DBMS_SESSION.SET_IDENTIFIER` instead of calling the procedure `DBMS_SESSION.CLEAR_IDENTIFIER`.

Accounting by Module and Action

An Oracle SQL trace file contains no information whatsoever on the response time of a module. All the information that the MERITS Profiler is able to distill from SQL trace files is inferred and requires extensive bookkeeping using complex hierarchical data structures. When a module and action are entered and left is derived from the instrumentation entries. A zero-length identifier is considered as an invalid module name. Hence no accounting per module and action occurs unless a module name is set. A zero-length identifier is considered as a valid action name and will show up as action name “NULL” in Profiler reports. If the profiler were to encounter the following two lines in a SQL trace file:

```
*** MODULE NAME:(SQL*Plus) 2007-02-06 15:53:20.844
*** ACTION NAME:() 2007-02-06 15:53:20.844
```

then the report would include a section on module and action “SQL*Plus.NULL”.

Module and action combinations that do not bracket any database calls are ignored. Since module and action are set on different lines the profiler evaluates each type of instrumentation entry separately.

Invocations of a subroutine may occur at different call depths represented by the field “dep” in PARSE, EXEC, and FETCH database calls. If a database client calls a subroutine directly, then it is invoked at dep=0. If the same subroutine is invoked from within a DML trigger on a table then the invocation will typically happen at dep=1.

Certain statistics of a database call are rolled up on higher call depth levels. They are CPU usage (c), elapsed time (e), consistent reads (cr), current blocks accessed (cu), and physical reads (p). Since those values are already rolled up by the DBMS engine they must be taken from the highest call depth within a module. On the contrary, the remaining statistics on the number of rows processed (r) and cursor misses (mis) are not rolled up by the DBMS engine. The Profiler stores them per call depth level and when a module is left it aggregates them from the highest to the lowest dep level. Of course it is not known ahead of time what the lowest call depth within a module will be. Keeping this algorithm in mind it is crucial that instrumentation is coded correctly. A task must begin and end at the same call depth level. Let’s assume that a task is not closed upon return from a subroutine but instead within the caller of the subroutine at a call depth that is one less than the one of the subroutine. The Profiler must assume that the resources consumed by any recursive statements at higher call depths are rolled up into one or more statements at lower call depths. If the end

task instruction is missing in a subroutine, the next statement after the subroutine call will happen at a higher dep level, but it will not account for any resources consumed within the subroutine, since it was started after the subroutine call was already complete. Hence the resource consumption by module and action cannot correctly state the resource consumption of the subroutine.

Guidelines for Instrumentation

As discussed in the previous section a stack based approach to instrumentation must be taken. For any programming language, a library should be available that provides the begin task and end task routines and implements a stack of module and action names. Essentially a begin task call is a push on a stack and an end task call is a pop off a stack. Currently such a library is only available for PL/SQL. The library for PL/SQL is called “Instrumentation Library for Oracle” (ILO) and the source code is available at no charge on Sourceforge (http://sourceforge.net/projects/ilo). ILO can be used from any programming language that supports calling PL/SQL packages.

Installation

The MERITS Profiler is distributed as a downloadable Zip archive. The software is ready to run once the Zip archive has been unpacked. Administrator privilege on Windows or root privilege on UNIX is not necessary to deploy the software. Simply choose a directory for unpacking the Zip archive and unpack it using unzip or a graphical tool such as the Windows Explorer. Below is a sample contents list of a Profiler distribution.

```
 Archive:  mprof-0.9.11.zip
   Length      Date    Time    Name
 ---------  ---------- -----   ----
         0  05/15/2010 16:15   mprof/
         0  12/08/2009 18:57   mprof/bin/
      1922  12/04/2009 18:13   mprof/bin/mprof
      2152  12/08/2009 18:30   mprof/bin/mprof.bat
         0  03/27/2011 17:28   mprof/conf/
      1266  03/27/2011 17:28   mprof/conf/css.properties
       824  03/27/2011 17:28   mprof/conf/oacdty_type_name_map.properties
       857  03/27/2011 17:28   mprof/conf/oct_map.properties
      2107  03/27/2011 17:28   mprof/conf/profiler.properties
      1642  03/27/2011 17:28   mprof/conf/profiler_log4j.properties
      5868  03/27/2011 17:28   mprof/conf/profiler_props_desc.properties
     20221  03/27/2011 17:28   mprof/conf/rt_queries.properties
         0  12/04/2009 18:15   mprof/contrib/
     11531  07/04/2009 00:31   mprof/contrib/apache_license.txt
     76539  08/13/2009 21:55   mprof/contrib/otn-distribution-license.html
         0  03/27/2011 15:48   mprof/doc/
       410  03/27/2011 14:15   mprof/doc/MERITS-Profiler-Book.log
    152983  08/18/2009 22:48   mprof/doc/MERITS-Profiler-Introduction-and-Features.pdf
    427458  03/27/2011 15:48   mprof/doc/MERITS-Profiler-User-Guide.pdf
         0  12/04/2009 18:15   mprof/lib/
    829799  07/05/2009 19:03   mprof/lib/contrib.jar
    324530  03/27/2011 17:29   mprof/lib/mprof.jar
   1890499  08/13/2009 21:53   mprof/lib/ojdbc5.jar
         0  12/02/2009 19:09   mprof/license/
         0  05/15/2010 16:14   mprof/samples/
         0  03/08/2011 22:39   mprof/sql/
       605  03/08/2011 22:30   mprof/sql/create_user.sql
       604  03/08/2011 22:39   mprof/sql/db_directory.sql
       494  07/04/2009 17:53   mprof/sql/license_server_info.sql
      1057  07/04/2009 17:53   mprof/sql/mprof_lic_db_user.sql
      2911  03/08/2011 22:15   mprof/sql/profiler_role.sql
       738  03/08/2011 22:30   mprof/sql/settings.sql
      1381  03/08/2011 22:30   mprof/sql/set_statistics_level.sql
 ---------                     -------
   3758398                     33 files
```

After unpacking the software, certain environment variables need to be set in order to use the Profiler. Please refer to the subsequent section pertaining to your operating system.

Setting MERITS Profiler Environment Variables

The MERITS Profiler uses four environment variables. They are:

- MPROF_HOME
- MPROF_JAVA_HOME

- MPROF_JDBC_DRIVER
- MPROF_PROPERTIES

Among these environment variables, MPROF_HOME and MPROF_JAVA_HOME are mandatory. MPROF_JDBC_DRIVER and MPROF_PROPERTIES are optional.

MPROF_HOME is the Profiler installation directory, i.e. the directory where the Zip file containing the software was unpacked. The last component of the path name must be “mprof”. MPROF_JAVA_HOME is the installation directory of a Java 1.6 or newer run-time environment. The Profiler uses <MPROF_JAVA_HOME>/bin/java (or equivalent) to launch a Java virtual machine. MPROF_JDBC_DRIVER may be set to use a non-default JDBC driver. If this environment variable is not set, the Profiler uses the JDBC driver in `mprof/lib/ojdbc5.jar`. MPROF_PROPERTIES is discussed in detail in section Parameter Reference on page 20.

Windows

The profiler installation directory is set using the variable MPROF_HOME. MPROF_HOME needs to be set to the directory where the software was unpacked with "\mprof" appended. For example, if you unpacked the software in "C:\Program Files" then set MPROF_HOME as below:

```
set MPROF_HOME=C:\Program Files\mprof
set MPROF_JAVA_HOME=C:\Program Files\java\jre6
```

Next, add the full path of the directory where mprof.bat resides to the PATH variable. For example:

```
C:\>set PATH=%PATH%;"C:\Program Files\mprof\bin"
```

You may now run the Profiler in offline mode by typing `mprof` at a command prompt:

```
C:\> mprof help=true
```

To persist the change of the PATH variable and to save the two new environment variable settings, edit them at user or system level by navigating to CONTROL PANEL > SYSTEM > ADVANCED > ENVIRONMENT VARIABLES.

If you forget to set one of the mandatory environment variables, the script mprof.bat will report an error:

```
C:\home\ndebes\it\java\esqltrcprof\trunk> mprof
Error: required environment variable MPROF_HOME is not set.
```

UNIX/Linux

This section assumes that you are using the Korn or Bash shell. MPROF_HOME needs to be set to the directory where the software was unpacked with "/mprof" appended. In the example below, the software was unpacked in the directory /home/ndebes.

```
$ export MPROF_HOME=/home/ndebes/mprof
$ export MPROF_JAVA_HOME=/home/ndebes/jre1.6.0_16
$ export PATH=$PATH:$MPROF_HOME/bin
$ mprof help=true
```

Add the environment variable settings to your shell’s login script (e.g. .profile), such that the variables will be set on subsequent logins.

Obtaining, Installing, and Using a License

The complimentary version of the MERITS Profiler has a limitation on the file size of SQL trace files that it processes. The maximum file size is 1 MB (2^{10} bytes). The licensing model incorporates the maximum trace file size as one aspect that affects licensing costs. The license fee increases as the maximum SQL trace file size increases.

Real-time mode and other advanced MERITS Profiler features such as aggregation of non-sharable SQL statements that lack bind variables are also only available with a valid license file. You need to designate an Oracle instance within your network as a license server. The Profiler connects to an Oracle instance on the license server to verify the database ID and platform of your license. This database is called the license database. If license verification completes successfully, the Profiler sets the maximum trace file size encoded in the license file and activates other licensed features such as real-time mode.

Obtaining a License File

The license is bound to certain characteristics of a license server. Each time the Profiler is started with a license file present, it connects very briefly to the license server instance to verify that the characteristics stored in the license file indeed match those of the license server. The Profiler exits with an error message if the verification fails. Choose a production Oracle DBMS instance with a static IP address that will not be decommissioned in the near future as the license server. You will not be able to run the Profiler in real-time mode should the license server be decommissioned or otherwise become unavailable. In such a case you will need to contact the vendor to obtain a new license file.

The verification requires access to the V$ view V$DATABASE. A script is provided for creating a database user with the minimum privileges to accomplish this task.

To obtain a license file, three pieces of information must be provided to the vendor:

1. The IP address of a listener, that the license server instance registers with.

2. The database ID of the database mounted by the license server instance.
3. The name of the operating system platform.

Item one is determined by looking up the value of the HOST field in your listener configuration. If the value is a host name, then translate it to an IP address using nslookup or a hosts file[1]. In the example below, the listener uses the IP address assigned to the host name “dbserver”.

```
listener=
  (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST =dbserver)(PORT = 1521))
  )
```

Translating the host name to an IP address yields item 1 for the license file.

```
C:\> nslookup 192.168.1.144
Server:  your.dns.server
Address:  192.168.1.1

Name:    dbserver
Address:  192.168.1.144
```

Here, the IP address of the listener is `192.168.1.144.`

Items two and three are obtained by querying V$DATABASE. Please use the script license_server_info.sql in <MPROF_HOME>/sql to get the information. The script has the following contents:

```
spool license_server_info.txt
SELECT dbid || ', '''||platform_name||'''' AS license_server_info
FROM sys.v_$database;
spool off
```

Connect to the license server instance as a database user with access to V$DATABASE and run the script as shown below:

```
C:\> sqlplus system
SQL*Plus: Release 10.2.0.3.0 - Production on Sat Jul 4 08:27:31 2009
Copyright (c) 1982, 2006, Oracle.  All Rights Reserved.
Enter password:
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
SQL> @license_server_info
LICENSE_SERVER_INFO
--------------------------------------------------------------------------------
2870266532, 'Microsoft Windows IA (32-bit)'
```

This creates the spool file license_server_info.txt. To request a license file, send the contents of this file along with the listener IP address to the vendor.

License Server Setup

A valid license file that matches an available Oracle DBMS instance in your network is required to complete the steps outlined in this section. When you receive your license file, save it in the directory <MPROF_HOME>/license within the Profiler installation. A license file is a plain text file named license.lic. A sample license file is reproduced below:

```
lic_listener_ip=192.168.1.144
lic_db_id=2870266532
lic_db_platform=Microsoft Windows IA (32-bit)
licensee=customer
license_version=1
licensor=ORADBPRO
profiler_version=0.9.0
Expiration=2009-7-19
Signature=302D02140156AE8DCC479222B8691746B148F3831A6F32B50215008DA52495FD43BE5ED09ABA94A6419568
8940D62F
```

The license file contains the three pieces of information that characterize the license server (IP address, database ID, platform), the name of the licensee, an expiration date, and some further information. Setting up real-time mode consists of the following three steps:

1. Create a new database user or designate an existing one for license verification
2. Encrypt the password of the chosen database user
3. Save the database user name and encrypted password in a property file

1. The UNIX hosts file is /etc/hosts. Windows uses %SystemRoot%\system32\drivers\etc\hosts, where the environment variable SystemRoot usually has the value C:\WINDOWS.

Database User

It is recommended to create a new database user with the minimum set of privileges for license verification. If you prefer using an existing database user, make sure that it has the privileges required to query the V$ view V$DATABASE. The script <MPROF_HOME>/mprof_lic_db_user.sql is provided to create a new database user. This script is reproduced below:

```
-- recommended database user name, may be changed if desired
define lic_db_user=mprof_license
create user &lic_db_user identified externally account lock;
GRANT CREATE SESSION TO &lic_db_user;
GRANT SELECT ON sys.v_$database TO &lic_db_user;
-- use of the password commmand ensures that the password is neither visible on screen
-- nor transmitted in clear text over a network
password &lic_db_user
alter user &lic_db_user account unlock;
```

The suggested name of the new database user is “mprof_license”. Change the define statement at the beginning of the script if you prefer a different user name. Due to the GRANT on SYS.V_$DATABASE the script needs to be run with SYSDBA privileges.

Password Encryption

For convenient use of the Profiler, the encrypted password of the license database user is saved in a user-specific property file. Run the Profiler with the parameter encrypt set to true as indicated below:

```
C:\> mprof encrypt=true
```

The Profiler then asks for a password to encrypt. The password is not echoed to the display and needs to be typed twice for verification.

```
Enter password to encrypt:
Re-enter password:
```

As a result, the Profiler prints the encrypted password:

```
Encrypted password is: Kjme6jHsB4E=
```

The final step consists of saving the encrypted password and the license database user name in a property file. This is described in the next section.

Property File

A Java property file is a plain text file that contains key/value pairs. For your convenience, the Profiler accepts all parameters either as command line parameters or as a key/value pair in a property file. To use real-time mode, both the license database user name and encrypted password must be set. If you were to use the suggested user name and the encrypted password generated above, then the property file would look as below:

```
lic_db_user=mprof_license
lic_db_encrypted_passwd=Kjme6jHsB4E=
```

The Profiler uses the JDBC Thin driver to connect to the license server instance and to verify the license file. A JDBC Thin URL has the following format:

```
jdbc:oracle:thin:@<host/IP address>:<port>/<instance service name>
```

The listener IP address is taken from the license file. The port number is assigned with the parameter lic_listener_port. This parameter has a default value of 1521. Thus, you only need to set it if you use a different port. The third component of the JDBC URL is an instance service name. Pick one of the service names the license server instance has registered with the license server listener. The command lsnrctl services [listener_name] displays all services registered with a listener. Then set the service name with the parameter lic_listener_service in the property file. For example:

```
lic_listener_service=orcl.world
```

You are now ready to test the three parameter settings in your property file. Make sure that you have copied the file license.lic to <MPROF_HOME>/license. Assuming that you have already used the Profiler in offline mode, the command script for starting the Profiler has already been customized for your environment and it will be able to locate required jar files as well as the license file by searching the Java class path. To test your setup, start the Profiler by passing the name of your property file with the parameter “properties”. Let’s assume that your property file is called myprops.properties. You would then start the Profiler as below:

```
C:\> mprof properties=myprops.properties
```

The Profiler will then assemble a JDBC URL from the parameters in the property file (or default parameter values) and the licensing information. If the setup is correct, the Profiler will respond as shown below:

```
INFO  profiler: Trying to verify license file C:\program files\mprof\license.lic
INFO  profiler: License file integrity verified: true; days left: 365
INFO  profiler: License server verified: true: '2870266532'=='2870266532' and 'Microsoft Windows
IA (32-bit)'=='Microsoft Windows IA (32-bit)'
```

```
INFO  profiler: Real-time mode: false
ERROR profiler: No SQL trace file specified with parameter sql_trace_file (real-time mode:
false).
```

As is evident from the output above, the Profiler verified the license. Since no further parameters were supplied it terminated with an error after the license verification.

MERITS Profiler Real-Time Mode

Real-time mode requires a database user with privileges to query certain DBA_* views, V$ views, and a few database objects in the schema PERFSTAT.

You have two options:

1. Use a database user that has the role DBA.

2. Create a role that has the minimum set of privileges required by real-time mode.

Option 2 is recommended for security-sensitive environments, since it guarantees that the database user account for the Profiler does not have access to sensitive tables or views.

The default name of the role for option 2 is MPROF_ROLE. It may be changed by editing the script profiler_role.sql.

In order to set up a database for the least privilege approach, run the script profiler_role.sql as SYS. This script is located in <MPROF_HOME>/sql.

```
SQL> @sql/profiler_role.sql
```

The output from the script profiler_role.sql is not shown. Once the role MPROF_ROLE has been created and privileges assigned to it, you may create a database user and assign it the role.

```
SQL> CREATE USER mprof_user IDENTIFIED EXTERNALLY ACCOUNT LOCK;
User created.
SQL> GRANT CREATE SESSION TO mprof_user;
Grant succeeded.
SQL> GRANT mprof_role TO mprof_user;
Grant succeeded.
```

To finish, assign a password and unlock the new account:

```
SQL> PASSWORD mprof_user
Changing password for mprof_user
New password:
Retype new password:
Password changed
SQL> ALTER USER mprof_user ACCOUNT UNLOCK;
User altered.
```

MERITS Profiler Concepts

The next sections address some important concepts that are relevant to performance diagnosis based on extended SQL trace files and to the MERITS Profiler in particular.

Figure 1 shows the six main building blocks of the MERITS Profiler. They are:

1. A user interface—this can be a graphical user interface or a command line user interface.
2. A follow mode trace file reader—this component has the capability of reading a growing trace file in real-time while an active traced process still writes to the file in append mode.
3. A parser that understands the extended SQL trace file format.
4. An accounting component, that keeps track of the information the parser extracted from a trace file.
5. A correlation component that retrieves performance diagnostic data that is related to the data extracted by the parser from the target Oracle instance.
6. A renderer component that invokes a final correlation step when the Profiler stops reading from the trace file and generates an HTML or text report on the information held by the accounting component.

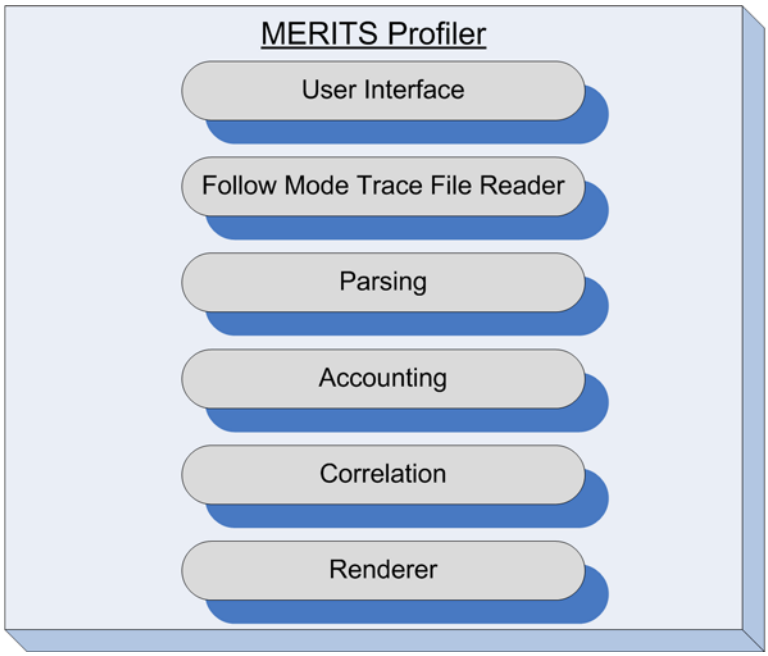


Figure 1: Components of the MERITS Profiler

Modes of Operation

The MERITS Profiler operates in two modes:

- offline mode
- real-time mode

Offline Mode

Offline mode is the mode of operation that you may be familiar with from other SQL trace profilers such as Oracle's TKPROF. A trace file is created using the SQL trace facility and some time after the trace file has been closed a profiler reads the trace file for analysis. This mode of operation does not require a license for the MERITS Profiler, but is severely limited in its diagnostic expressiveness. It is usually impossible to solve a performance problem by solely analyzing a SQL trace file. Much more information like initialization parameters, object statistics or results from querying V$ views may be needed. I call this lack of information from no diagnostic data but a SQL trace file the "diagnostic gap". Offline-mode results in an overly large diagnostic gap.

Real-Time Mode

In real-time mode, the MERITS Profiler reads, parses, and correlates a SQL trace in real-time, i.e. as soon as new data is appended to a SQL trace file, the Profiler reads it, parses it, and correlates it with V$ views or other relevant sources of diagnostic data available by accessing an Oracle DBMS instance.

Figure 2 illustrates the Profiler's real-time mode. An Oracle DBMS instance (1) is the target environment where a performance diagnosis occurs. A DBMS instance has a database (2) and the files comprising the database open. A traced process (3) writes the SQL and/or PL/SQL statements it executes to a SQL trace file. The traced process serves a database client (not shown). Tracing can

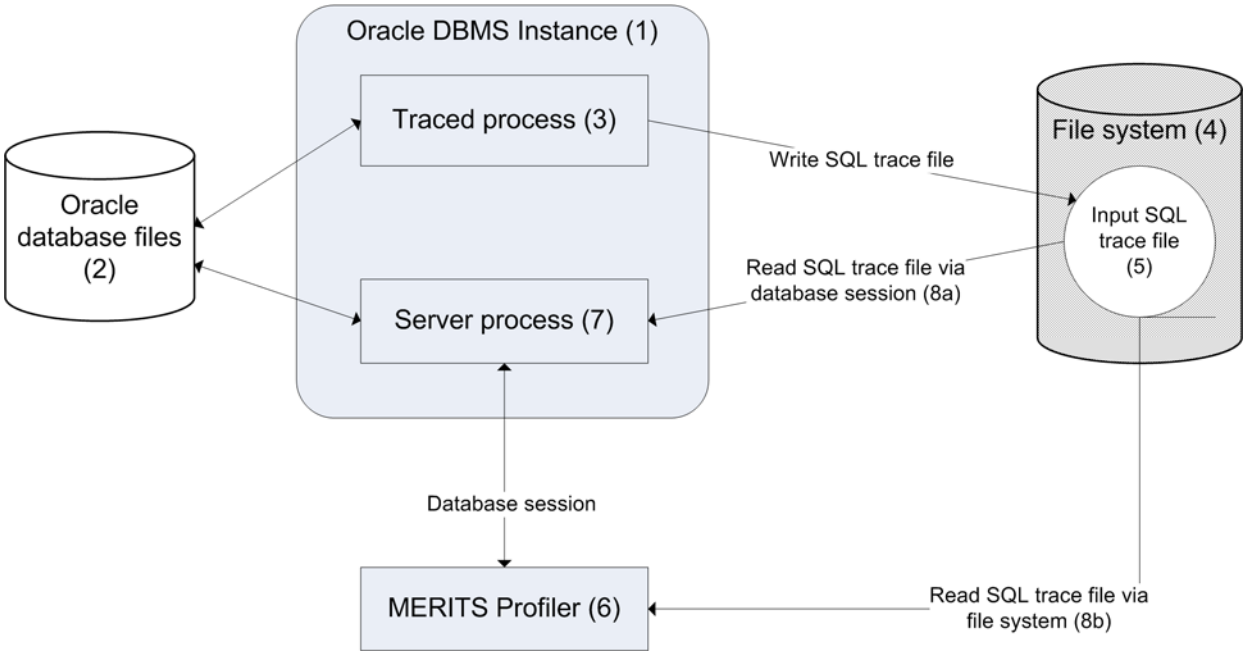


Figure 2: Schematic Overview of the MERITS Profiler's Real-Time Mode

be enabled in a variety of ways that are well known among Oracle database administrators (e.g. DBMS_SYSTEM, DBMS_SUPPORT). The MERITS Profiler can enable extended SQL trace for a session with the parameter "session" using the package DBMS_MONITOR. SQL trace files are always written to a file system (4). The MERITS Profiler supports two ways of reading an input SQL trace file (5):

1. as a file system file (8a)
2. via the database session to the target instance using a BFILE (8b)

Option 1) requires that the MERITS Profiler (6) runs on the same system as the traced process or that a network file system of some sort allows file system access to the trace file in cases where the Profiler does not run on the system where the traced process resides.

Option 2) requires a database directory object that points to the directory where the trace file resides.

In real-time mode, the MERITS Profiler (6) connects to the target instance (1) for the purpose of correlating data in a SQL trace file with data available from the Oracle DBMS instance (e.g. V$ views). Correlation queries the Profiler runs are processed by a server process (7) of the target instance.

Put simply, the MERITS Profiler (6) reads the SQL trace file (5), correlates it with information from the DBMS instance (1) and renders a performance diagnostic report.

Resource Profile

A resource profile is an apportionment of response time often presented in tabular form. Each row of a tabular *resource profile* contains a single contributor to response time, e.g. CPU time consumption or a *wait event*. Ideally, the following columns are included: contributor name, percentage of response time, contribution to response time in seconds, number of times the contributor was called upon, and average time consumed by a call upon the contributor. The table should be sorted by percentage of contribution to response time in descending order.

In a MERITS Profiler report, resource profiles tell the performance analyst where the response time went. There is a resource profile that covers an entire trace file at the beginning of a report. This is followed by individual resource profiles for each distinct SQL or PL/SQL statement.

Think Time

If a "SQL*Net message from client" wait event exceeds the think time threshold, the excess portion of the wait time is accounted for with the pseudo wait event think time in resource profiles. The term pseudo wait event is used, since this is a wait event name introduced by the Profiler. The Oracle DBMS has no knowledge of think time and does not have a wait event called "think time" The portion of "SQL*Net message from client" that equals the think time threshold, is reported under the original event name. As a consequence of introducing the pseudo wait event "think time", two wait events are generated for "SQL*Net message from client" waits that exceed the threshold. The resulting higher count of wait events has an effect on the average duration of the wait events

related to think time and also on the average duration of all contributors to response time. The total number of round-trips between client and server equals the number of times a “SQL*Net message from client” wait has been recorded.

The rationale behind the concept of think time is that a database client that does not make the next request on the database server in less than think_time_threshold_ms is engaging in non database-related work. Think time cannot be reduced by optimizing an Oracle database or instance. The contribution of think time to response time is important, since it limits the maximum speedup from optimization. Hence think time is reported as a separate contributor to response time. Think time is also crucial as proof that database access is not the cause of a performance problem.

Analyzing Trace Files

The next two sections contain examples of using the Profiler in offline and real-time modes.

Offline Mode

In offline mode, the Profiler does not connect to the target DBMS instance for correlating a SQL trace file with V$ views and other information available from the target Oracle instance. Furthermore the trace file must be read as a file system file. A license file is not required. Certain feature are disabled without a license. All required parameters may be given on the command line, however it is more convenient to create a file for setting parameters and to refer to the properties file with the environment variable `MPROF_PROPERTIES`. On Windows this can be done with the following command:

```
C:\> set MPROF_PROPERTIES=offline.properties
```

You might start by setting just the directory where the trace files reside and turning off interactive mode in a properties file. Note that you need to escape the backslashes in Windows path names with another backslash as shown below:

```
C:\> type offline.properties
trace_file_directory=c:\\oracle\\product\\admin\\ten\\udump
interactive=false
```

If you do not have a license file, you're ready to run the Profiler:

```
C:\> mprof sql_trace_file=ten_ora_2984.trc
INFO  profiler: License file 'license.lic' not found in
CLASSPATH=C:\Oracle\product\db10.2\jdbc\lib\ojdbc14.jar;C:\program files\mprof\lib\contrib.
jar;C:\program files\mprof\license;..
Real-time mode and aggregation of non-reusable statements by MD5 hash value are disabled.
INFO  profiler: Real-time mode: false
INFO  profiler: Starting thread 8 for input file 'ten_ora_2984.trc' in file system directory
'c:\oracle\product\admin\ten\udump'
INFO  parser: Stop attempting to read trace file since total_sleep_time=0 >= max_idle_time=0 and
interactive_mode=false
INFO  profiler: Finished generating report '.\report.html'.
```

If you do have a license file, you need to make sure that the license file is in the `CLASSPATH`. The recommended location for the license file `license.lic` is `<MPROF_HOME>/license`, since this directory is added to the class path in the wrapper script that starts the Profiler (`mprof.bat` on Windows). Aggregation of non-sharable SQL statements with varying literals instead of bind variables is enabled in offline mode, if a license file is present. The lic_* parameters must be specified, such that the Profiler can verify the license server. A sample properties file for that purpose is shown below:

```
C:\> type offline.properties
trace_file_directory=c:\\programme\\oracle\\product\\admin\\ten\\udump
interactive=false
lic_listener_service=ten.oradbpro.com
lic_db_user=mprof_license
lic_db_encrypted_passwd=H8Q/1pIXlb+0NCOHyDqlKQ==
```

Assuming that the environment variable `MPROF_PROPERTIES` is not set, you would call the profiler as below:

```
C:\> mprof properties=offline.properties sql_trace_file=ten_ora_2984.trc
INFO  profiler: Trying to verify license file C:\program files\mprof\license\license.lic
INFO  profiler: License file integrity verified: true; days left: 3670
INFO  profiler: License server verified: true: '2870266532'=='2870266532' and 'Microsoft Windows
IA (32-bit)'=='Microsoft Windows IA (32-bit)'
INFO  profiler: Real-time mode: false
INFO  profiler: Starting thread 8 for input file 'ten_ora_2984.trc' in file system directory
'c:\oracle\product\admin\ten\udump'
INFO  parser: Stop attempting to read trace file since total_sleep_time=0 >= max_idle_time=0 and
interactive_mode=false
INFO  profiler: Finished generating report '.\report.html'.
```

Real-time Mode

This section contains a very small example of running the MERITS Profiler in real-time mode. The case study shows how to use the MERITS Profiler to trace a session based on its SID and SERIAL# and how to generate a report. First of all you should create a properties file for setting all the properties required in real-time mode. An example properties file `user_props.properties` that contains all the properties for the target as well as the license server Oracle DBMS instance is reproduced below. Run the Profiler with the parameter encrypt=true to generate encrypted passwords for the license server and target databases. Remember that the encryption is valid only for the operating system user that generated it. You cannot use an encrypted password that was created by another operating system user. The parameters `jdbc_url` and `db_encrypted_passwd` both refer to the target instance and data-

base. All the lic_* parameters refer to the license server database. Since lic_listener_port is not set, the default of 1521 is used.

```
jdbc_url=jdbc:oracle:thin:@localhost:1521/TEN.oradbpro.com
db_encrypted_passwd=Kjme6jHsB4E=
lic_listener_service=ten.oradbpro.com
lic_db_user=mprof_license
lic_db_encrypted_passwd=H8Q/1pIXlb+0NCOHyDqlKQ==
```

Once you have a property file like the one above, all you need to do to run the Profiler in real-time mode is to pass the parameter db_user on the command line.

Next, start a database client (SQL*Plus in this case) and connect to the target Oracle DBMS instance, then get the SID and SERIAL# for the session from V$SESSION.

```
C:\> sqlplus example
SQL*Plus: Release 10.2.0.4.0 - Production on Wed Aug 5 16:52:13 2009
Copyright (c) 1982, 2006, Oracle.  All Rights Reserved.
Enter password:
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - Production
SQL> select sid,serial# from v$session where sid=userenv('sid');
       SID    SERIAL#
---------- ----------
        47        203
```

Start the MERITS Profiler and use SID.SERIAL# from the above query result as the value for the parameter session. This parameter tells the profiler to enable SQL trace on the specified session. The file user_props.properties contains the parameters required for connecting to the target database.

```
C:\home\ndebes\it\java\esqltrcprof\trunk>mprof properties=user_props.properties
db_user=mprof_user session=47.203 report_name=ins_customer.html
INFO  profiler: Trying to verify license file
C:\home\ndebes\it\java\esqltrcprof\trunk\license\license.lic
INFO  profiler: License file integrity verified: true; days left: 3670
INFO  profiler: License server verified: true: '2870266532'=='2870266532' and 'Microsoft Windows
IA (32-bit)'=='Microsoft Windows IA (32-bit)'
INFO  profiler: Real-time mode: true
INFO  profiler: Automatically determined trace file name 'ten_ora_2984.trc'
INFO  profiler: Automatically determined database directory name 'USER_DUMP_DEST' for DBMS
instance parameter user_dump_dest='C:\PROGRAMME\ORACLE\PRODUCT\ADMIN\TEN\UDUMP'
INFO  profiler: Starting thread 8 for input file 'ten_ora_2984.trc' in database directory
'USER_DUMP_DEST'
Choose:
1. Generate Report and Exit
2. Status
3. Abort
4. Disable SQL Trace
WARN  parser: Trace file ten_ora_2984.trc could not be read (java.io.IOException:
java.sql.SQLException: ORA-22288: Datei- oder LOB-Vorgang GETLENGTH nicht erfolgreich
Das System kann die angegebene Datei nicht finden.
; error code=22288); retrying ...
WARN  parser: Trace file ten_ora_2984.trc could not be read (java.io.IOException:
java.sql.SQLException: ORA-22288: Datei- oder LOB-Vorgang GETLENGTH nicht erfolgreich
Das System kann die angegebene Datei nicht finden.
; error code=22288); retrying ...
```

Next, execute an INSERT statement, a COMMIT, and a SELECT:

```
SQL> variable empno number
SQL> INSERT INTO EMP (ename,job) VALUES ('Debes', 'DBA') RETURNING empno INTO :empno;
1 row created.
SQL> COMMIT;
Commit complete.
SQL> SELECT * FROM emp WHERE empno=:empno;
     EMPNO ENAME      JOB              MGR HIREDATE                   SAL
---------- ---------- --------- ---------- ------------------- ----------
      COMM     DEPTNO
---------- ----------
      9204 Debes      DBA
```

The trace file gets created and the MERITS Profiler's follow mode reader automatically starts reading the trace file.

```
WARN  parser: Warning: Execution plan (STAT entry) for cursor 4 without prior PARSING IN CURSOR #4
is ignored (line 25)
WARN  parser: Warning: Execution plan (STAT entry) for cursor 4 without prior PARSING IN CURSOR #4
is ignored (line 26)
WARN  parser: Warning: Execution plan (STAT entry) for cursor 4 without prior PARSING IN CURSOR #4
is ignored (line 27)
INFO  parser: EOF reading trace file - sleeping for 1 s ...
INFO  parser: EOF reading trace file - sleeping for 2 s ...
INFO  parser: EOF reading trace file - sleeping for 4 s ...
INFO  parser: EOF reading trace file - sleeping for 8 s ...
INFO  parser: EOF reading trace file - sleeping for 1 s ...
INFO  parser: EOF reading trace file - sleeping for 2 s ...
INFO  parser: EOF reading trace file - sleeping for 4 s ...
```

Choose menu item 2 for displaying the Profiler’s program status:

```
2
Thread 8: state: TIMED_WAITING; parsing trace file; 13310 bytes read from ten_ora_2984.trc (EOF
encountered 2 time(s))
Choose:
1. Generate Report and Exit
2. Status
3. Abort
4. Disable SQL Trace
```

As you can tell from the screen output above, the Profiler had reached an end of file condition on the input trace file twice. Finally choose menu item 1 for generating the report and to quit the MERITS Profiler.

```
1
INFO  rt: Reporting object statistics for TABLE NDEBES.EMP
INFO  profiler: Finished generating report '.\ins_customer.html'.
```

At this point a report in HTML format has been generated.

Parameter Reference

This section contains detailed descriptions of the MERITS Profiler's parameters in alphabetical order. All the parameters in this section may be passed on the command line. Alternatively all parameters except user_properties may be specified using a Java properties file. There are two options for specifying a full or relative path to a properties file with user-specific settings:

- The command line parameter user_properties, which is not supported inside a property file.
- The environment variable MPROF_PROPERTIES.

Using MPROF_PROPERTIES is more convenient than passing user_properties=<path> each time the Profiler is run. An example for Windows follows. The environment variable MPROF_PROPERTIES may be set on the command line as below:

```
C:\> set MPROF_PROPERTIES=C:\user_props.properties
```

Alternatively, MPROF_PROPERTIES may be set permanently as a user environment variable via Control Panel > System > Advanced > Environment Variables. Assuming that the file C:\user_props.properties contains a valid trace file directory path using the parameter trace_file_directory, merely the parameter sql_trace_file needs to be passed on the command line:

```
C:\> mprof.bat sql_trace_file=ten_ora_8144_spreport.trc
```

When called in this way without setting any further parameters, the Profiler writes a log file called profiler.log and generates an HTML report in a file called profiler.html.

awr_flush_level

Attribute	*Description*
Data type	string
Default value	OFF
Range of values	OFF, TYPICAL, ALL

The parameter awr_flush_level controls the setting of FLUSH_LEVEL in calls of the Active Workload Repository (AWR) package function DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT (see *Oracle Packages and Types Reference* manual). The meanings of the permissible settings are:

- OFF: no AWR snapshot is taken
- TYPICAL: FLUSH_LEVEL TYPICAL is used
- ALL: FLUSH_LEVEL ALL is used

If awr_flush_level has a value other than OFF, the MERITS Profiler creates an AWR snapshot when it connects to the target DBMS instance and another AWR snapshot when the user requests a report. The Profiler automatically creates an AWR and ASH report for the interval between the two snapshots. The parameter awr_flush_level is ignored when the Profiler is running in offline mode, i.e. when the parameter db_user is not set. Note that use of AWR and ASH requires a license for the Diagnostics Pack from Oracle Corporation or one if its subsidiaries.

cached_table_threshold_mb

Attribute	*Description*
Data type	integer
Default value	100
Range of values	-1, 1 to 2^{31}-1

The parameter cached_table_threshold_mb specifies the threshold for a segment's size (in MB) beyond which it is included in the report section on buffer cache contents. A value of -1 instructs the Profiler to omit the report section entitled "Buffer Cache Contents".

date_format

Attribute	*Description*
Data type	string
Default value	%1$td-%1$tb-%1$ty %1$tT
Range of values	Any value that complies with the format specification of the Java class java.util.Formatter.

The parameter date_format specifies the date format for Profiler reports. Each element must start with %1$, since all the formatting specifications apply to a single Java variable.

db_directory

Attribute	*Description*
Data type	string
Default value	none
Range of values	Name of a database directory object that points to the directory designated with the initialization parameter USER_DUMP_DEST or any other directory containing SQL trace files.

The parameter db_directory instructs the Profiler to read a SQL trace file using the Oracle DBMS's BFILE feature. The names of directory objects are case-sensitive. Read permission on the directory objects used must be available. The SQL syntax for granting read permission on a directory is:

```
GRANT READ ON DIRECTORY <db_directory> TO <grantee>;
```

The parameters db_directory and trace_file_directory are mutually exclusive.

db_encrypted_passwd

Attribute	*Description*
Data type	string
Default value	none
Range of values	Valid encrypted password

The parameter db_encrypted_passwd specifies the encrypted password for the database user on the target database. Run the Profiler with the parameter encrypt=true to encrypt clear-text passwords.

db_release

Attribute	*Description*
Data type	integer
Default value	none
Range of values	10, 11

The parameter db_release specifies the Oracle DBMS major release (e.g. '10') that generated a SQL trace file. It must be specified if a trace file does not contain a trace file header with release information. The string may contain only digits.

db_user

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid Oracle database user name

The parameter db_user specifies a database user name for connecting to a target Oracle DBMS instance. The parameter db_user must be set in order to enable correlations in real-time mode. Setting db_user is also required when using one of the following parameters:

- session
- awr_flush_level
- sp_snap_level

encrypt

Attribute	*Description*
Data type	boolean
Default value	false
Range of values	true or false

The parameter setting encrypt=true instructs the Profiler to encrypt a clear-text password. The Profiler prompts twice for a password without echoing the password typed. Then it displays the encrypted password on the screen. To enhance security, the encryption key depends on the operating system user name that is in effect while the Profiler runs. Hence, encrypted passwords cannot be shared among different operating system users. If encrypt=true, the Profiler encrypts a password and exits.

help

Attribute	*Description*
Data type	boolean
Default value	false
Range of values	true or false

Use the parameter setting help=true to see an overview of the Profiler’s parameters.

interactive

Attribute	*Description*
Data type	boolean
Default value	true
Range of values	true or false

The parameter interactive controls whether or not the Profiler displays a menu for interacting with the user. Interactive mode allows the user control when the trace file analysis ends. It is useful when tracing a certain code path in an interactive application. The setting interactive=true cannot be used with a max_idle_time setting that is greater than zero. If interactive=true, the Profiler displays

a menu that gives access to the status of the trace file analysis. The setting interactive=false is appropriate when the trace file to be analyzed is no longer growing, since the process that generated it has terminated or tracing was disabled.

jdbc_url

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid JDBC Thin URL with the following format: jdbc:oracle:thin:@<host>:<port>/<instance service name>

JDBC URL for connecting to a target Oracle instance in real-time mode.

lic_db_encrypted_passwd

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid encrypted password

The parameter lic_db_encrypted_passwd specifies the encrypted password for the database user on the license server database. Run the Profiler with the parameter encrypt=true to encrypt clear-text passwords.

lic_db_user

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid Oracle database user name

The parameter lic_db_user specifies a database user name on the license server database. The Profiler connects to the license server Oracle instance for license validation.

lic_listener_port

Attribute	*Description*
Data type	integer
Default value	1521
Range of values	1024 to 65535

The port number of the Oracle TNS Listener that is used to connect to the license server Oracle DBMS instance.

lic_listener_service

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any instance service name that the license server Oracle DBMS instance registered with the license server TNS Listener.

The parameter lic_listener_service specifies the instance service name for connecting to the license server Oracle DBMS instance. The Profiler uses this setting for license validation.

log4j_config_url

Attribute	*Description*
Data type	string
Default value	A platform-dependent file URL that points to the file profiler_log4j.properties in <MPROF_HOME>/conf
Range of values	Any valid file URL that points to a Java properties file for Apache log4j.

The parameter log4j_config_url specifies the name of a property file that contains the log4j configuration. A file URL that represents an absolute path must be used when overriding the default value. File URLs have platform-specific syntax. File URLs on Windows contain the drive letter, e.g. file:///C:/somedir/my_log4j.properties). UNIX file URLs contain “file:” followed by an absolute path.

log4j_pattern_layout

Attribute	*Description*
Data type	string
Default value	%p %c: %m%n
Range of values	Any valid Apache log4j layout specification

The parameter log4j_pattern_layout specifies a layout for logging with Apache log4j. Available placeholders are: t: thread, p: log level, c: logger, F: file, L: line number, m: message, n newline character. Each placeholder must be preceded by a percent sign (%).

logfile

Attribute	*Description*
Data type	string
Default value	profiler.log
Range of values	Relative or absolute platform-specific path name for the Profiler log file

The parameter logfile specifies the log file used by the Profiler. The format of log file entries is controlled by the parameter log4j_pattern_layout.

max_bind_sections

Attribute	*Description*
Data type	integer
Default value	10
Range of values	1 to 2^{31}-1

The parameter max_bind_sections specifies the maximum number of bind sections to include in a Profiler report. Bind variables are important when running a captured SQL statement in order to reproduce its response time. A trace file may contain thousands of bind variable sections, which in turn may contain dozens of bind variables. Use max_bind_sections to control how many bind variable sections are incorporated into a report.

max_idle_time

Attribute	*Description*
Data type	integer
Default value	0
Range of values	1 to 2^{31}-1

The parameter max_idle_time specifies the maximum time in seconds to wait for a SQL trace file to grow or to be created in non-interactive mode (interactive=false). If a trace file that is analyzed in real-time does not grow or is not created when the parameter session is used within max_idle_time, the Profiler stops trying to read the trace file. If more than 0 bytes could be read from the trace file the Profiler will attempt to create a report.

max_statements

Attribute	*Description*
Data type	integer
Default value	2000
Range of values	1 to 2^{31}-1

The parameter max_statements specifies the maximum number of distinct statements to include in a report. The purpose of this parameter is to avoid large reports containing statements with negligible contributions to response time.

mod_act_max_statements

Attribute	*Description*
Data type	integer
Default value	50
Range of values	0 to 2^{31}-1

The parameter mod_act_max_statements specifies the maximum number of distinct statements per module and action to include in a report. The purpose of this parameter is to avoid large listings of statements invoked by a module that have negligible contributions to response time. Setting mod_act_max_statements=0 disables the subsection entitled “Invoked Statements” within the section “Results by Module and Action” entirely.

object_statistics

Attribute	*Description*
Data type	boolean
Default value	true
Range of values	true or false

The parameter object_statistics controls whether or not the Profiler retrieves object statistics on tables and indexes in real-time mode and incorporates the result in a report.

output_directory

Attribute	*Description*
Data type	string
Default value	. (current directory)
Range of values	Any valid directory with write permission

The parameter output_directory is used for specifying where the Profiler writes reports.

properties

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid path to a file that conforms to the specification of a Java properties file.

The parameter properties is intended for retrieving reusable parameter settings from a Java properties file in a file system accessible to the Profiler. Parameter settings made by a properties file overrule default parameter values. Parameters passed to the Profiler on the command line overrule parameter settings in a Java properties file.

Note: use the environment variable MPROF_PROPERTIES to refer to customized parameter settings without having to pass the parameter properties on the command line.

real_time

Attribute	*Description*
Data type	boolean
Default value	true
Range of values	true or false

Use the parameter real_time to enable or disable real-time correlations between a SQL trace file and data dictionary or V$ dynamic performance views. The setting real_time=true is ignored unless the parameter db_user is set. Real-time correlations require a database session, hence db_user must be set to honor real_time=true. If db_user is not set, the Profiler disables real-time correlations and informs the user as shown below:

```
INFO  profiler: Real-time mode: false (db_user='')
```

Note that trace files may be read as BFILEs over a database session when db_user is specified and real_time=false. In other words trace files may be transferred using a database session without using the database session for performing correlations. The parameters sql_trace_level, awr_flush_level, and sp_snap_level are honored when real_time=false. Of course db_user must be set to use the aforementioned parameters.

report_name

Attribute	*Description*
Data type	string
Default value	report.html
Range of values	Any valid file name without a directory component.

Use the parameter report_name to specify an output file name for a Profiler report.

session

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid combination of <SID>.<SERIAL#> that corresponds to a session that is currently connected to the target DBMS instance.

Use the parameter session to instruct the Profiler to enable SQL trace on a database session using the PL/SQL package DBMS_MONITOR and to begin analyzing the resulting SQL trace file. See parameter sql_trace_level for controlling the SQL trace level. The parameter session is ignored unless the parameter db_user is also set. The profiler attempts to automatically determine the trace file name as well as a database directory for reading a trace file as a BFILE through the LOB (**L**arge **OB**jects) API. Hence there is no need to use the parameters db_directory and sql_trace_file if the parameter session is set.

sp_snap_level

Attribute	*Description*
Data type	integer
Default value	7
Range of values	-1, 0, 5, 6, 7, 10

Use the parameter sp_snap_level to instruct the Profiler to take Statspack snapshots at the indicated level (the value -1 indicates that no Statspack snapshots shall be taken). The first snapshot is taken just after the Profiler connects to the target DBMS instance. The second snapshot is taken when the Profiler is asked to create a report or when it automatically stops reading the trace file in non-interactive mode (interactive=false). If the parameter session is also set, the Profiler passes the session ID as an argument to the function STATSPACK.SNAP, such that Statspack captures diagnostic data for the traced session in addition to the instance-level diagnostic data.

sql_trace_file

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid file name of a SQL trace file without a directory component.

Use the parameter sql_trace_file to specify the input SQL trace file to be processed by the Profiler.

sql_trace_level

Attribute	*Description*
Data type	integer
Default value	12
Range of values	-1, 1, 4, 8, 12

The parameter value -1 indicates that the SQL trace level of the session specified using the parameter session is to remain unaltered. The other permissible values of the parameter control the SQL trace level in the same way as the level of the undocumented event 10046[1]. Table 2 shows the relationship between SQL trace levels and the kind of trace file entries generated at a particular level. A database call is a generic term for a parse, execute, or fetch call. The value of the parameter sql_trace_level is ignored

Table 2: SQL Trace Levels

SQL Trace Level	*Database Calls*	*Bind Variable Values*	*Wait Events*
1	yes	no	no
4	yes	yes	no
8	yes	no	yes
12	yes	yes	yes

unless the parameters db_user and session are also set. In interactive mode, a menu offers the user the choice to disable SQL trace before generating a report and leaving the Profiler application or to keep SQL trace enabled beyond the execution of the Profiler. The latter option may be useful to get an intermediate result for a long-running session. The menu displayed in interactive mode is shown below:

```
$ mprof session=34.60 sql_trace_level=12
Choose:
1. Generate Report and Exit
2. Status
3. Abort
4. Disable SQL Trace
```

If the Profiler is started in non-interactive mode with sql_trace_level>0, then it will automatically disable SQL trace when it reaches the end of the trace file and max_idle_time has been exceeded. Thus, the Profiler may be used to enable SQL trace and to automatically create a report when the trace file has no longer grown during an interval specified with max_idle_time.

1. See “Secret ORACLE” by Norbert Debes, Lulu Enterprises Inc., 2008, ISBN 978-1-4357-0551-7

statistics_level

Attribute	*Description*
Data type	string
Default value	ALL
Range of values	ALL, TYPICAL

The Profiler parameter statistics_level is used to specify a value for the Oracle initialization parameter by the same name. If the value of the Profiler parameter statistics_level differs from the current value of the parameter statistics_level used by the target DBMS instance, the Profiler will temporarily adjust the DBMS parameter accordingly. This functionality is available in real-time mode only. Setting statistics_level=ALL while gathering performance diagnostic data makes sure that SQL statement execution statistics can be retrieved with DBMS_XPLAN.DISPLAY_CURSOR. Execution plans obtained at statistics_level=ALL contain estimated vs. actual row counts and allow easy identification of inaccurate estimates by the cost based optimizer (CBO).

think_time_threshold_ms

Attribute	*Description*
Data type	integer
Default value	5
Range of values	1 to 60000

The parameter think_time_threshold_ms sets the threshold in milliseconds. If a “SQL*Net message from client” wait exceeds the think time threshold, the excess portion of the wait time is accounted for with the pseudo wait event think time. The remainder, which equals the think time threshold, is reported under the original event name “SQL*Net message from client”.The rationale behind this classification is that a database client that does not make the next request on the database server in less than think_time_threshold_ms is engaging in non database-related work. Think time cannot be reduced by optimizing an Oracle database or instance. The contribution of think time to response time is important, since it limits the maximum speedup from optimization. Hence think time is reported as a separate contributor to response time. Think time is also crucial as proof that database access is not the cause of a performance problem.

trace_file_directory

Attribute	*Description*
Data type	string
Default value	none
Range of values	Any valid file system directory.

Use the parameter trace_file_directory to inform the Profiler about the directory where the input trace file resides. This parameter cannot be used in conjunction with db_directory. When used in a properties file, backslashes in the path name must be escaped with the backslash character (\) since they have special meaning. To access trace files in `C:\traces`, you need to set:

```
trace_file_directory=C:\\traces
```

in a properties file.

use_awr

Attribute	*Description*
Data type	boolean
Default value	false
Range of values	true or false

The parameter use_awr controls whether or not the Profiler attempts to perform correlations between an analyzed SQL trace file and AWR views in real-time mode. If use_awr=true the Profiler retrieves exeuction statistics and plans pertaining to statements in an analyzed SQL trace file from AWR views. Note that accessing any of the AWR DBA_HIST_* views requires a license for the Diagnostics Pack from Oracle Corporation or one of its subsidiaries.

use_statspack

Attribute	*Description*
Data type	boolean
Default value	false
Range of values	true or false

The parameter use_statspack may be used to enable or disable correlations between an analyzed trace file and Statspack repository tables in real-time mode. Note that Statspack is not installed by default. You should not set use_statspack=true unless a working installation of Statspack exists in the schema PERFSTAT of the target database. If use_statspack=true the Profiler attempts to retrieve execution plans captured by Statspack pertaining to SQL statements in the analyzed trace file. Contrary to the Active Workload Repository (AWR) Statspack is free of charge.

Profiler Report Structure and Contents

This section provides detailed information on the structure and contents of a MERITS Profiler report. A report contains resource profiles, histograms, a trace file header, and so on. Many sections such as the one containing optimizer statistics, table columns, indexes, and buffer cache contents appear only if the Profiler is used in real-time mode with a license file. Hence the report sections for real-time mode and offline mode are documented in two separate sections.

Offline Mode

The subsequent sections describe Profiler report sections that appear in offline as well as real-time mode. Certain sections contain additional information in real-time mode. Items that appear only in real-time mode are marked as such.

Report Date

This section contains the point in time when the report was generated and the Profiler version that generated the report.

```
Report Date: Fri 04-Dec-09 17:10:15 (generated by MERITS Profiler release 0.9.6)
```

Abbreviations

This section explains the meanings of abbreviations used throughout reports.

Trace File Header

An Oracle DBMS trace file contains a header with interesting information such as the number of CPUs in the server, the amount of memory, the point in time when the trace file was created, etc. The header starts at line one of a trace file and continues up to the first line that starts with three asterisks.

Note that the trace file header is missing if a trace file was created using Oracle's TRCSESS utility or when the session level parameter `trace_file_identifier` was changed by the database client after tracing had begun. If there is no trace file header in a trace file, the Profiler cannot extract the DBMS version from the header and the parameter db_release must be set to inform the Profiler about the Oracle DBMS release.

Sessions

This section lists the session identifiers encountered in a trace file. The format is <sid>.<serial> where sid corresponds with the column SID in V$SESSION and serial corresponds with the column SERIAL# in V$SESSION. Both numbers combined uniquely identify a database session within the lifetime of an Oracle instance, i.e. the period between an instance startup up and the subsequent shutdown. Session identifiers are not unique throughout the lifetime of a database. Due to the limitations outlined in section *Limitations* on page 48 the analysis of trace files containing more than a single session may lead to incorrect results. Below is an example report section showing a session identifier:

```
Session in trace file(s): 1
40.9
```

Response Time and Statistics

This section contains statistics and a resource profile for a trace file as a whole. In other words this section includes data derived from all the cursors found in a trace file. Below is an example:

```
Accounted-for response time (R): 1963.814 s
Measurement interval (delta tim): 1969.648 s
Measurement interval based on converted tim: 24-Jul-09 18:48:09.171 to 24-Jul-09 19:20:58.820
First timestamp: 24-Jul-09 18:48:09.203
Last timestamp: 24-Jul-09 19:20:58.812
Intra Database Call Wait Time: 316.344 s
Inter Database Call Wait Time: 11.071 s
Committed transactions: 20676
Committed transactions/s (based on delta tim): 10.50
Transaction rollbacks: 0
Commits (read only): 14738
Rollbacks (read only): 0
Oracle PARSE ERROR entries: 0
MERITS Profiler parser errors: 1014
Resource Profile for Trace File C:\traces\ten_ora_14552-swingbench.trc
```

Response Time Contributor	Duration (s)	Percent (%)	Count	Average (s)
CPU (PARSE, EXEC, FETCH, CLOSE)	994.562500	50.5	207643	0.004790
unknown	647.684631	32.9	27	23.988317
db file sequential read	315.864838	16.0	26437	0.011948
SQL*Net message from client	8.847790	0.4	14737	0.000600
think time	2.127134	0.1	79	0.026926
log file switch completion	0.245403	0.0	5	0.049081
read by other session	0.123776	0.0	6	0.020629
SQL*Net message to client	0.095734	0.0	14736	0.000006
buffer busy waits	0.061340	0.0	31	0.001979
latch: cache buffers chains	0.022697	0.0	17	0.001335
log file sync	0.009506	0.0	31	0.000307
latch free	0.001054	0.0	7	0.000151
enq: TX - row lock contention	0.001030	0.0	9	0.000114
enq: TX - contention	0.000621	0.0	2	0.000311
latch: In memory undo latch	0.000345	0.0	4	0.000086
enq: FB - contention	0.000028	0.0	2	0.000014
cursor: pin S	0.000004	0.0	1	0.000004
Total	1969.648315	100.0	263774	0.007467

Statistics

Table 3 provides detailed information on the statistical metrics seen in the example above.

Table 3: Statistical Metrics

Metric	*Meaning*
Accounted-for response time (R)	The response time derived from the instrumentation of the Oracle DBMS. R is defined as the sum of the elapsed time in PARSE, EXEC, and FETCH calls at recursive call depth zero plus the elapsed time of inter database call wait time.
Measurement interval (delta tim)	Delta tim is the difference between the highest and the lowest tim value in a trace file. Most trace file entries contain the parameter tim, which is a timestamp with microsecond resolution. Essentially delta tim is wall-clock elapsed time whereas R is accounted for elapsed time. As such, delta tim is more reliable than R. By comparing the accounted-for response time with delta tim you may assess the instrumentation quality of the DBMS server. Ideally the trace file entries would account for almost 100% of delta tim. If there is a large discrepancy, then the trace file probably records functionality that is poorly instrumented, such as LOB or SecureFile access. Another scenario is when CPU resources on the database server are so scarce that a substantial amount of elapsed time is not accounted for by trace file entries. This causes a large contribution of "unknown" origin.
Intra Database Call Wait Time	Wait time that is caused by a database call. For example if a FETCH call requires a disk read, a db file sequential read or db file scattered read wait event contributes to the elapsed time of the FETCH.
Inter Database Call Wait Time	Inter database call wait time is the wait time accounted for by the events SQL*Net message to client and SQL*Net message from client.
Committed transactions	Derived from XCTEND entries where rlbk=0 and rd_only=0.
Committed transactions/s (based on delta tim)	This figure is calculated by dividing the number of committed transactions by delta tim.
Transaction rollbacks	Derived from XCTEND entries where rlbk=1 and rd_only=0.
Commits (read only)	Derived from XCTEND entries where rlbk=0 and rd_only=1.
Rollbacks (read only)	Derived from XCTEND entries where rlbk=1 and rd_only=1.
Oracle PARSE ERROR entries	The number of PARSE ERROR entries due to incorrect syntax or insufficient privileges to access one or more database objects.
MERITS Profiler parser errors	The number of lines in a trace file that the MERITS Profiler could not parse successfully. This number is always greater than zero since the Profiler does not recognize all the information in platform-specific trace file headers. If a trace file contains not just SQL trace data but also systemstate or errorstack dumps then several thousand Profiler parser errors are to be expected. Severe parser errors, such as when a WAIT event entry or database call cannot be parsed are reported both in the terminal window as well as in a log file.

Resource Profile

A resource profile is an apportionment of response time often presented in tabular form. Each row of a tabular resource profile contains a single contributor to response time, e.g. CPU consumption or a wait event. The following columns are included: contributor name, contribution to response time in seconds (column heading "Duration"), percentage of response time (column heading "Percent"), number of times the contributor was called upon (column heading "Count"), and average time consumed by a call upon the contributor (column heading "Average"). The data in a resource profile is sorted by contribution to response time in descending order. Hence the most significant contributors to response time appear in the top lines of the table. Reducing their impact will yield the most benefit on the overall reduction of response time through tuning.

The last line of a resource profile contains totals for all columns. The total duration is identical to delta tim. MERITS Profiler resource profiles contain up to two synthetic response time contributors. They are “unknown” and “think time”. A detailed explanation of think time is on page 15. The difference between delta tim and CPU time plus wait time is shown as “unknown”. If a large portion of the response time is attributed to an unknown origin, then either the trace file captured Oracle DBMS functionality that is poorly instrumented or CPU resources on the DBMS server were so scarce that elapsed time increased significantly since DBMS server processes had to wait for a CPU to become available. The example above is from a scenario with scarce CPU resources. A large portion of the difference between delta tim and CPU time plus wait time is not explained by wait events and is reported as “unknown”.

Database Call Statistics

At trace file level, this section shows totals and averages based on the three types of database calls (PARSE, EXEC, FETCH) in the first three lines. The column “Count” is derived by counting the database calls in a trace file. The values in column “Elapsed” are derived from the parameter e (elapsed time) of a database call trace file entry. The values in column “CPU” are derived from the parameter c of a database call trace file entry. Averages are obtained by dividing the elapsed or CPU times by the number of database calls in the column “Count”.

The values in the column “Disk” are derived from the database call parameter p (physical reads). Logical reads are represented by the values in column “Query”. These are derived from the database call parameter cr (consistent reads). The column entitled “Block Changes” shows values derived from the database call parameter cu (current blocks). Values in column “Rows” are derived from the parameter r (rows). Library cache misses are found in the column entitled “Cursor Misses”. They are derived from the database call parameter mis (miss).

DB Call	Count	Elapsed (s)	Average Ela. (s)	CPU (s)	Average CPU (s)	Disk	Query	Block Changes	Rows	Cursor Misses
PARSE	14814	0.391	0.000026	0.641	0.000043	0	0	0	0	0
EXEC	132520	1952.352	0.014733	993.922	0.007500	26437	6816008	330369	58924	0
FETCH	60309	0.000	0.000000	0.000	0.000000	0	0	0	1084772	0
Total	207643	1952.744	0.009404	994.563	0.004790	26437	6816008	330369	1143696	0
Avg. per EXEC	1.567	0.014735	n/a	0.007505	n/a	0.199	51.434	2.493	8.630	0.000
Avg. per FETCH	3.443	0.032379	n/a	0.016491	n/a	0.438	113.018	5.478	18.964	0.000
Avg. per Row	0.182	0.001707	n/a	0.000870	n/a	0.023	5.960	0.289	1.000	0.000

Buffer cache hit ratio: 99.6 %
Parse call library cache hit ratio: 100.0 %

The bottom three lines of the database call statistics provide additional averages per execution (“Avg. per EXEC”), per fetch call (“Avg. per FETCH”), and per row retrieved by the database client (“Avg. per Row”). By looking at the value where the table row “Avg. per FETCH” meets the column “Rows” it becomes evident how many rows are retrieved on average by a FETCH database call. It is much more efficient to retrieve a large number of rows in large batches of say 100 instead of small batches of five or less. Each access to a block including repeated accesses to the same block when the amount of prefetched rows is less than the number of rows per block counts as a consistent read (parameter cr). The higher the position of a column within the ordered sequence of columns of a table, the higher the CPU cost incurred by accessing the column. Putting frequently accessed columns at the beginning of a column list in a CREATE TABLE statement saves CPU time during FETCH calls.

By looking at the value where the table row “Avg. per Row” meets the column “Query” it becomes evident how many consistent reads are needed to retrieve a single row. Values lower than ten are considered good. High values may indicate a full table scan or a large index range scan.

Two additional metrics are provided along with the database call statistics. They are the buffer cache hit ratio and the parse call library cache hit ratio. The buffer cache hit ratio is calculated according to the formula below:

$$\left(1 - \frac{p}{cr + cu}\right) \cdot 100$$

In the equation above, the variables *p*, *cr*, and *cu* are the sum of the values of the database call parameters by the same name over all database calls.

The parse call library cache hit ratio is calculated using this formula:

$$\left(1 - \frac{mis}{PARSE\ calls}\right) \cdot 100$$

In the above formula the variable *mis* is the total number of PARSE calls with mis=1 and the variable *PARSE calls* represents all the PARSE calls irrespective of the parameter value of mis.

Elapsed Time, CPU Usage, and Wait Time by Recursive Call Depth

This section shows whether the statements sent by the database client are responsible for the majority of response time or whether recursive statements triggered by the aforementioned statements are the main issue. The data is shown in tabular form. Each column holds the data for a single recursive call depth. The number of columns depends on how many different recursive call depth levels are present in the trace file analyzed. Three table rows are used to differentiate among database call elapsed time (row heading “DB Call Elapsed Time”; derived from the parameter e of database call entries), CPU time (row heading “CPU Time”; derived

from the parameter c of database call entries), and wait time (row heading “Wait Time”; derived from the parameter ela of WAIT entries).

Recursive Call Depth	0	1	2
DB Call Elapsed Time (s)	140.708 (1952.743)	1811.460 (1812.035)	0.575
CPU Time (s)	33.687 (994.563)	960.750 (960.875)	0.125
Wait Time (s)	8.953 (325.274)	315.823 (316.321)	0.498

The values in parentheses are cumulative values that are aggregated over higher levels of recursive call depth. If the database client calls PL/SQL routines that in turn execute many SQL statements, then it is likely that the SQL statements executed at a higher recursive call depth are responsible for the majority of the response time. The example above is from such a scenario. The database call elapsed time at recursive call depth 1 is 1811.460 s whereas the value for recursive call depth 0 is only 140.708 s. Hence the majority of the elapsed time is caused by SQL or PL/SQL statements at recursive call depth 1.

LOB Operation Statistics

Oracle11*g* SQL trace includes metrics for LOB operations. LOB operations cannot be attributed to a SQL statement. Hence LOB operation statistics cannot be included in the section on individual statements. Currently LOB operations are only reported at the trace file level.[1]

LOB Operation Statistics

Operation	Calls	Elapsed (s)	CPU (s)	Disk	Query	Blk. Chngs.
LOBFILCLOSE	0	0.000000	0.000000	0	0	0
LOBFILOPN	0	0.000000	0.000000	0	0	0
LOBGETLEN	0	0.000000	0.000000	0	0	0
LOBREAD	12374	0.426126	0.020000	25	25	0
LOBTMPCREATE	8772	0.100896	0.000000	0	0	0
LOBTMPFRE	8772	0.394375	0.000000	0	0	43860
LOBWRITE	8772	30.208376	26.150000	0	0	61404
Total	38690	31.129772	26.170000	25	25	105264

Results for Individual Statements

This section contains metrics for all the distinct SQL statements in a trace file. It starts with a section entitled “Top Statements”.

Top Statements

The “Top Statements” section contains only statements executed at recursive call depth zero (dep=0). Service time is the response time spent in the DBMS server plus IPC latency wait time. Thus, service time excludes think time. IPC latency wait time is the total wait time of the events “SQL*Net message from client” that did not exceed the think time threshold plus the total “SQL*Net message to client” wait time associated with a certain statement.

Service time is the metric used to rank the top statements. The statement response time R includes think time. The percentage of statement contribution to response time is reported based on delta tim. Statements at ranks higher than 10 that contribute less than 1.0% to the total service time are not listed. The statement text is truncated to at most 80 characters. An example “Top Statements” section is reproduced below:

Top Statements

Rank	Hash Value/Cursor	SQL ID	Service Time (s)	Percent	R (s)	Percent	Statement Text
1	966758382		1410.635	71.62 %	1411.207	71.65 %	BEGIN :1 := orderentry.neworder(:2,:3,:4); END;
2	1631089791		188.788	9.58 %	189.252	9.61 %	BEGIN :1 := orderentry.browseproducts(:2,:3,:4); END;
3	3589721925		160.440	8.15 %	160.875	8.17 %	BEGIN :1 := orderentry.newcustomer(:2,:3,:4,:5,:6,:7); END;
4	4030344732		139.015	7.06 %	139.318	7.07 %	BEGIN :1 := orderentry.browseandupdateorders(:2,:3,:4); END;
5	2086907756		62.413	3.17 %	63.160	3.21 %	BEGIN :1 := orderentry.processorders(:2,:3); END;
6	8		0.000	0.00 %	0.002	0.00 %	Note: statement text unavailable due to absence of parse call from trace file
Total			1961.290	99.58 %	1963.814	99.70 %	

In real-time mode, the Profiler attempts to determine the SQL ID for statements in Oracle10g trace files and includes the SQL ID as an additional column in the “Top Statements” section. The SQL ID is present in Oracle11g trace files and is hence included in the “Top Statements” section even in offline mode.

If SQL trace is enabled after some statements have already been parsed and these statements are reused, there is no PARSING IN CURSOR entry for those statements in the trace file. Hence the statement text and hash value (as well as the SQL ID) associated with a certain cursor are not available. Such statements are listed under their cursor number as shown in the example below:

Rank 26: Cursor 27

1. Oracle bug 12589689 entitled “NEED TO INCLUDE LOB LOCATOR IN TRACE FILES, CURRENTLY WE HAVE ONLY LOBREAD” contains an enhancement request to make LOB operations attributable to SQL statements.

```
~~~~~~~~~~~~~~~~~~

Response Time: 0.023 s (0.00 % of delta tim)
Recursive Call Depth: 1
Module: 'New Customer'
Action: 'getOrdersByCustomer'
Waits: 3
Total Wait Time: 0.023 s
Intra Database Call Wait Time: 0.023 s
Line Number: 1078

Statement Text
~~~~~~~~~~~~~~

Note: statement text unavailable due to absence of parse call from trace file
```

The hash value and statement text of cursors that are associated with a PARSING IN CURSOR entry is known. Both items are included in the report. The format used is as shown in the following example:

```
Rank 2: Statement with Hash Value 2863564559
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Response Time: 1081.732 s (54.92 % of delta tim)
SQL ID: dw2zgaapax1sg
Force Matching Signature: 3138049466602010507
MD5 Hash Value: 3A7DCCD2344B7F8D7C3C39C1B56E88C5
Command Type: 3 (SELECT)
Parsing User ID: 78 (SOE)
Parsing Schema ID: 78 (SOE)
Optimizer Goal: 1 (ALL_ROWS)
Recursive Call Depth: 1
Module: 'New Order'
Action: 'getProductDetailsByCategory'
Waits: 30
Total Wait Time: 0.026 s
Intra Database Call Wait Time: 0.026 s
Line Number: 110

Statement Text
~~~~~~~~~~~~~~

SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME, PRODUCT_DESCRIPTION, CATEGORY_ID, WEIGHT_CLASS,
WARRANTY_PERIOD, SUPPLIER_ID, PRODUCT_STATUS, LIST_PRICE, MIN_PRICE, CATALOG_URL,
QUANTITY_ON_HAND FROM PRODUCTS, INVENTORIES WHERE PRODUCTS.CATEGORY_ID = :B1 AND
INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID ORDER BY INVENTORIES.WAREHOUSE_ID
```

The metrics available for individual statements are explained in Table 4.

Table 4: Metrics for Individual Statements

Metric	*Meaning*
Response Time Including Think Time	The sum of all e values plus the sum of all inter database call waits (i.e. “SQL*Net message from client” and “SQL*Net message to client”).
Response Time Excluding Think Time	The same as “Response Time Including Think Time” above, except that only “SQL*Net message from client” wait time below the think time threshold is considered.
Recursive Elapsed Time	The total elapsed time associated with recursive statements triggered by the current statement.

Table 4: Metrics for Individual Statements

Metric	*Meaning*
SQL ID	The SQL identifier associated with the statement text. The SQL ID is part of the SQL trace file format starting with Oracle DBMS release 11.1. The profiler attempts to determine the SQL ID based on the hash value in Oracle10g trace files by accessing the V$ view V$SQL in real-time mode.
Force Matching Signature	The force matching signature from the V$ view V$SQL. Statements with identical force matching signatures are candidates for cursor sharing if the initialization parameter cursor_sharing has a non-default value. This item is present in real-time mode only.
MD5 Hash Value	A hash value calculated based on the normalized statement text using the MD5 algorithm. Normalization replaces literals with placeholders, such that statements that differ only by literals have the same MD5 hash value. The MERITS Profiler includes a parser for SQL statements. The purpose of the MD5 hash value is to identify statements that differ only by literals. The profiler automatically aggregates statements with identical MD5 hash values. Hence the impact of statements that are semantically identical but are not reusable due to varying literals is shown under a single statement instead of as hundreds or thousands of different statements that may contribute only marginally to response time. Unfortunately software vendors build Oracle database applications without bind variables. The MD5 hash value aids with performance diagnoses of such applications.
Command Type	The type of SQL or PL/SQL command, e.g. SELECT, UPDATE, DELETE, etc. The command name is reported in addition to the numeric command type found in the trace file.
Parsing User ID	The numeric user ID of the database user that parsed the statement. This is either the user ID of the user that logged in or 0 (SYS). In real-time mode the numeric ID is resolved to the user name. If the resolution fails, e.g. because the user no longer exists in the database, “n.r” for not resolvable is reported.
Parsing Schema ID	The numeric schema ID of the schema that provided the context for parsing the statement. In real-time mode the numeric ID is resolved to the user name. If the resolution fails, e.g. because the user no longer exists in the database, “n.r” for not resolvable is reported.
Optimizer Goal	The numeric optimizer goal found in the trace file. The denominator of the optimizer goal (e.g. ALL_ROWS, CHOOSE) is shown in parentheses.
Recursive Call Depth	The recursive call depth (dep) of the first PARSING IN CURSOR entry associated with the current statement.
Waits	The total number of WAIT entries associated with a cursor.
Total Wait Time	The total wait time associated with a cursor.
Intra Database Call Wait Time	The total intra database call wait time associated with a cursor.
Inter Database Call Wait Time	The total inter database call wait time associated with a cursor. “SQL*Net message from client” and “SQL*Net message to client” are classified as inter database call wait events. All other wait events are intra database call wait events.
IPC latency wait time	The total wait time of “SQL*Net message from client” below the think time threshold plus the total wait time of “SQL*Net message to client” associated with a cursor.
Line Number	The line in the trace file where a statement (or cursor) was first encountered.

Statement Level Resource Profile and Database Call Statistics

The sections entitled “Statement Level Resource Profile” and “Database Call Statistics” at statement level have the same structure as at trace file level. Please refer to the information on these sections outlined earlier.

Row Prefetch Histogram

A row prefetch histogram provides additional information on FETCH database calls associated with the execution of SELECT statements. A FETCH call may prefetch a certain number of rows from a database table or some other result set. The number of prefetched rows is called “fetch array size” by some sources. The value of the parameter r in FETCH SQL trace file entries shows how many rows have been fetched.

The SQL*Plus parameter ARRAYSIZE controls how many rows are prefetched by SQL*Plus. A JDBC application may modify the default prefetch size by calling the method OracleConnection.setStatementCacheSize(). Generally each Oracle DBMS interface has its own method for modifying the prefetch size.

If the number of prefetched rows is less than the number of rows in a database block, the same block has to be visited by multiple FETCH database calls causing unnecessary consistent reads. Retrieving a large result set of several hundred or even thousands of rows is much more efficient with a large prefetch size. A row prefetch histogram is very useful for identifying FETCH database calls that use too small a prefetch size. FETCH calls are assigned to histogram buckets based on the number of rows fetched. For example, if a FETCH call returned 15 rows it is assigned to the bucket for the range “<= 16” since 15 is larger than 8 and smaller than 16. The smallest range is “<=1”. Each subsequent range covers twice the prefetch size of the previous range.

Inefficient processing is characterized by a large number of rows retrieved with small prefetch sizes. Most or all of the elapsed time of a SELECT statement might be in the bucket for the range “<= 1”. For each histogram bucket that is responsible for a certain range of rows retrieved with a single fetch call, the following additional information is provided:

- how many rows were retrieved by all the FETCH calls in the histogram bucket (column Rows)
- the average number of rows retrieved by a single FETCH call (column Avg. Rows)
- the elapsed time of all the FETCH calls in the histogram bucket (column Elapsed)
- the percentage of elapsed time of all the FETCH calls in a bucket vs. the total elapsed time of all FETCH calls represented by a histogram (column Pct. Elapsed)
- the average elapsed time of all the FETCH calls in a bucket (column Avg. Ela.)
- the number of FETCH calls assigned to a bucket (column Fetch Calls)
- the percentage of FETCH calls in a bucket vs. the total number of all the FETCH calls represented by a histogram

An example of a row prefetch histogram is reproduced below. The database client retrieved 195 rows with an average prefetch size of 15 rows. All together 15 FETCH calls were made and 86.7 % of the FETCH calls had an average prefetch size of 15 rows.

```
Row Prefetch Histogram
----------------------
```

Range	Rows	Avg. Rows	Elapsed (s)	Pct. Elapsed	Avg. Ela. (s)	Fetch Calls	Percent
<= 1	1	1.0	0.007907	71.5	0.007907	1	6.7
<= 2	0	0.0	0.000000	0.0	0.000000	0	0.0
<= 4	0	0.0	0.000000	0.0	0.000000	0	0.0
<= 8	8	8.0	0.000160	1.4	0.000160	1	6.7
<= 16	195	15.0	0.002999	27.1	0.000231	13	86.7
Total	204	13.6	0.011066	100.0	0.000738	15	100.0

Recursive Descendants

This section contains the cursor number or hash value for each recursive statement triggered by the current statement. An excerpt from the statement text is also shown. The column “Parsing ID” is used to differentiate among recursive statements that are part of the application code and recursive statements that are part of the Oracle DBMS code base. The latter statements have a parsing ID value of 0 (SYS). In real-time mode the Parsing ID is resolved to the user name, if possible.

Hash Value	Elapsed Time (s)	Parsing ID	Statement Text (Excerpt)
2863564559	1068.692	78 (SOE)	SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME, PRODUCT_DESCRIPTIO
2084491117	67.198	78 (SOE)	SELECT CUSTOMER_ID, CUST_FIRST_NAME, CUST_LAST_NAME, NLS_LAN
2637862082	39.020	78 (SOE)	SELECT QUANTITY_ON_HAND FROM PRODUCT_INFORMATION P, INVENTOR

Execution Plan

This section contains the execution plans associated with a cursor (if any). Multiple execution plans per statement are recognized and reported. Except under rare circumstances or when tracing a tuning session with varying optimizer settings within a single session, the execution plan for a statement remains constant.

```
Execution Plan 1 (1 occurrence(s)):
-----------------------------------
```

ID	PID	Rows	Operation	Ela. (Self)	Ela. (Cum.)	CR (Self)	CR (Cum.)	PR (Self)	PR (Cum.)	PW (Self)	PW (Cum.)
1	0	4409	TABLE ACCESS BY INDEX ROWID ORDER_ITEMS	0.816097	25.307964	4506	45147	68	1915	0	0
2	1	8818	NESTED LOOPS	0.169614	24.491867	0	40641	0	1847	0	0
3	2	4409	NESTED LOOPS	0.103506	22.706345	0	31699	0	1728	0	0

```
   4   3      4409    TABLE ACCESS BY INDEX ROWID ORDERS       0.335824     0.479326      4584     14063        24        25         0         0
   5   4      4409     INDEX RANGE SCAN ORD_STATUS_IX          0.143502     0.143502      9479      9479         1         1         0         0
   6   3      4409    TABLE ACCESS BY INDEX ROWID CUSTOMERS   12.727343    22.123512      4409     17636       955      1703         0         0
   7   6      4409     INDEX UNIQUE SCAN CUSTOMERS_PK          9.396168     9.396168     13227     13227       748       748         0         0
   8   2      4409   INDEX RANGE SCAN ORDER_ITEMS_PK           1.615911     1.615911      8942      8942       119       119         0         0
```

Physical Reads by Database Object

This section reports physical reads by database object. The database object ID is derived from the parameter obj# in wait events related to physical reads (“db file sequential read” and “db file scattered read”). The meanings of the column headings are explained in Table 5.

Table 5: Physical Reads

Metric	*Meaning*
Obj. ID	Object ID (obj#); In real-time the object ID is resolved to segment owner and name using the data dictionary view DBA_OBJECTS.
SB Reads	Single block reads
SBR Time	Single block read time; The total elapsed time of all single block reads pertaining to a certain database object.
Avg. SBR Time	Average single block read time.
MB Reads	Multi-block reads, i.e. the number of “db file scattered read” wait events pertaining to a certain segment.
MBR Time	Multi-block read time; The total elapsed time of all multi-block reads pertaining to a certain database object.
Avg. MBR Time	Average multi-block read time.
MBR Blocks	The total number of blocks read using multi-block reads.
Avg. Blocks	The average number of blocks read in a single “db file scattered read” wait event, i.e. a multi-block read operating system call. Note that blocks that are already in the buffer cache are not read again when a segment is accessed using multi-block reads. The more blocks of a segment are already in the cache, the less likely it will be that the DBMS can use the full multi-block read size configured with the initialization parameter db_file_multiblock_read_count.
Owner & Segment	Owner and name of the segment. This column is present in real-time mode only.

Below is an example of a section with detailed data on physical reads:

```
Obj. ID  SB Reads    SBR Time Avg. SBR Time  MB Reads    MBR Time Avg. MBR Time MBR Blocks Avg. Blocks Owner & Segment
_______ _________ _________ _____________ _________ _________ _____________ __________ ____________ ____________________________________
  19155         1       0.016         0.016         1       0.001         0.001          5          5.0 HR.JOBS
  19139         1       0.008         0.008         1       0.001         0.001          5          5.0 HR.EMPLOYEES
  19162         1       0.000         0.000         1       0.001         0.001          5          5.0 HR.LOCATIONS
  19138         1       0.000         0.000         0       0.000         0.000          0          0.0 HR.DEPT_LOCATION_IX
  19136         1       0.000         0.000         0       0.000         0.000          0          0.0 HR.DEPARTMENTS
```

In the above example, one single block read and 1 multi-block read occurred on the segment HR.JOBS. The multi-block read consisted of five blocks read in a single read request to the operating system. It took 1 ms to perform the five block multi-block read.

Buffer Busy Waits

This section provides detailed information on buffer busy waits. It may be helpful in identifying frequently accessed blocks. Only blocks that were involved in buffer busy waits of more than 1 ms are reported.

```
Buffer Busy Waits with Wait Time >= 1 ms
_______________________________________

File & Block      Count Wait Time (s)
_____________ ________ _____________
7.11854               2         0.002
```

Use the following query to resolve the file and block to the segment owner and name:

```
SQL> SELECT owner, segment_name
```

```
FROM dba_extents
WHERE file_id=&file_id
AND &block_id BETWEEN block_id and block_id + blocks -1;
Enter value for file_id: 7
old   3: WHERE file_id=&file_id
new   3: WHERE file_id=7
Enter value for block_id: 11854
old   4: AND &block_id BETWEEN block_id and block_id + blocks -1
new   4: AND 11854 BETWEEN block_id and block_id + blocks -1

OWNER    SEGMENT_NAME
-------- ---------------
SOE      ORDERS
```

Note that the above query may be quite expensive. Hence it is not run automatically by the Profiler.

Captured Bind Variables

This section contains a configurable number of bind variable sets. The bind variables were supplied to the DBMS server as actual values for placeholders in SQL or PL/SQL statements by an application. The contents of this section are derived from BINDS trace file entries. The number of bind variable sets included in a report is limited using the parameter max_bind_sections.

```
Bind Section 1:
...............

Position Type              Value
________ ________________  ____________________________________________________
       0 null              null
       1 NUMBER            1029951
       2 NUMBER            0
       3 NUMBER            0


Bind Section 2:
...............

Position Type              Value
________ ________________  ____________________________________________________
       0 null              null
       1 NUMBER            1039626
       2 NUMBER            0
       3 NUMBER            0
```

Results by Module and Action

The Profiler is able to attribute database calls and wait events to modules and actions. An instrumented Oracle client application informs the DBMS engine about the tasks it performs by setting module and action. The section “Results by Module and Action” is where the benefits from instrumenting software may be reaped. This section points out which module and action is responsible for the highest resource consumption.

Module and action combinations are sorted based on delta tim within the trace file section(s) attributed to the respective module and action. For a module and action, delta tim is the difference between the first tim value encountered after entering a module and action combination and the last tim value before leaving a module and action.

Each section for a module and action contains three subsections:

- A resource profile
- Database call statistics
- Statements invoked while the module and action combination was in effect

Resource usage of the invoked statements is reported including the resource usage of recursive descendants. Please refer to the section on “Individual Statements” to find the resource consumption by recursive descendants alone.

```
Results by Module and Action
____________________________

Distinct combinations of module and action: 5

Rank 1: Module and Action 'img_load.lob_load'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Calls: 100
```

```
Delta tim: 5.299378 s
Accounted for R: 1.850351 s

Response Time Contributor                          Duration (s) Percent (%)     Count Average (s)

SQL*Net message from client                            1.649443        89.1     11300    0.000146
CPU (PARSE, EXEC, FETCH, CLOSE)                        0.234375        12.7       120    0.001953
SQL*Net vector data from client                        0.207704        11.2     11200    0.000019
db file sequential read                                0.039569         2.1       213    0.000186
SQL*Net message to client                              0.028717         1.6     11300    0.000003
SGA: allocation forcing component growth               0.022815         1.2         1    0.022815
enq: TX - contention                                   0.002560         0.1         1    0.002560
direct path write                                      0.002423         0.1         4    0.000606
enq: HW - contention                                   0.001559         0.1         1    0.001559
Disk file operations I/O                               0.000312         0.0         1    0.000312
unknown                                               -0.339126       -18.3         1   -0.339126
Total                                                  1.850351       100.0     34142    0.000054

Database Call Statistics
------------------------

DB Call             Count   Elapsed (s) Average Ela. (s)       CPU (s) Average CPU (s)  Disk  Query  Curr. Blocks  Rows Cursor Misses

PARSE                   5         0.000         0.000000         0.000        0.000000     0      0             0     0             0
EXEC                  105         0.172         0.001640         0.234        0.002232     0      0             0     1             0
FETCH                   5         0.000         0.000000         0.000        0.000000     0      0             0     1             0
CLOSE                   5         0.000         0.000008         0.000        0.000000     0      0             0     0             0
Total                 120         0.172         0.001435         0.234        0.001953     0      0             0     2             0
Avg. per EXEC       1.143      0.001640              n/a      0.002232             n/a 0.000  0.000         0.000 0.019         0.000
Avg. per FETCH     24.000      0.034438              n/a      0.046875             n/a 0.000  0.000         0.000 0.400         0.000
Avg. per Row       60.000      0.086096              n/a      0.117188             n/a 0.000  0.000         0.000 1.000         0.000

Number of invoked statements: 8

Rank Hash Value/Cursor SQL ID        Execs DB Calls Elapsed (s) CPU (s)     Disk Query Curr. Blocks Rows Csr. Mis. Statement Text

   1                 0 null              0        0       1.656   0.000        0     0            0    0         0 Note: statement text unavailable due
   2        1776478125 6rpu5mxny5txd   100      100       0.194   0.234        0     0            0    0         0 CALL BEGIN_TASK(:module, :action, :c
   3        3096556448 0kkhhb2w93cx0     1        3       0.004   0.000        0     3            1    1         0 update seg$ set type#=:4,blocks=:5,e
   4         429649310 1rts790ctrvcy     1        4       0.000   0.000        0     1            0    0         0 select lobj# from lobcomppart$ where
   5        1570213724 bsa0wjtftg3uw     1        5       0.000   0.000        0     3            0    1         0 select file# from file$ where ts#=:1
   6        4125516341 9gkq7rruycsjp     1        4       0.000   0.000        0     1            0    0         0 select parttype, partcnt, partkeycol
   7        1429416219 b1rhz2jam6a8v     1        4       0.000   0.000        0     1            0    0         0 select parentobj# from sys.lobfrag$
   8        2812718187 537c8ufmudb3b     0        0       0.000   0.000        0     0            0    0         0 SELECT /* OPT_DYN_SAMP */ /*+ ALL_RO
```

Wait Event Histograms

This section provides histogram data for each wait event encountered in a trace file. Wait events are assigned to histogram buckets based on the duration of the wait event. Each histogram bucket covers wait events with a certain minimum and maximum duration. The maximum wait time associated with a bucket doubles from one bucket to the next. Unused buckets at the low end are not displayed. In the subsequent example, the lowest bucket covers wait events that took longer than 8 µs and at most 16 µs. According to the table column “Count” two wait events were assigned to this histogram bucket. According to the column “Percent” these two wait events account for 22.2% of the total elapsed time caused by the wait event. Columns that contain the abbreviation “Cum.” (for cumulative) in their headings contain cumulative values for the respective bucket and all buckets with a smaller range. The value 66.7 in the column with the heading “Pct. Cum. Count” (row 4; column 10) means that 66.7 % of the wait events represented by the histogram completed in 128 µs or less. Cumulative values for a certain row are calculated by aggregating the column values of all the rows with a smaller range.

```
Histogram for Wait Event 'enq: TX - row lock contention'
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        Range  Elapsed (s) Pct. Elapsed        Count Percent Avg. Ela. (s) Cum. Ela. (s) Pct. Cum. Ela.   Cum. Count Pct. Cum. Count

  <= 16 micros    0.000023          2.2            2    22.2      0.000012      0.000023            2.2            2            22.2
  <= 32 micros    0.000000          0.0            0     0.0      0.000000      0.000023            2.2            2            22.2
  <= 64 micros    0.000000          0.0            0     0.0      0.000000      0.000023            2.2            2            22.2
 <= 128 micros    0.000391         38.0            4    44.4      0.000098      0.000414           40.2            6            66.7
 <= 256 micros    0.000351         34.1            2    22.2      0.000175      0.000765           74.3            8            88.9
 <= 512 micros    0.000265         25.7            1    11.1      0.000265      0.001030          100.0            9           100.0
        Total     0.001030        100.0            9   100.0      0.000114
```

Real-Time Mode

The subsequent sections describe Profiler report sections that appear solely in real-time mode. Note that only additional report sections are generated in real-time mode. There are no report sections that appear in offline mode but are omitted in real-time mode.

Active Workload Repository Snapshots

If the MERITS Profiler parameter awr_flush_level has the value typical or all, then the Profiler takes an AWR snapshot after it connects to the target DBMS instance and before it starts parsing the input trace file. It takes another AWR snapshot when the input trace file is exhausted or the user instructs the Profiler to stop reading the input trace file. The snapshot numbers are included at the beginning of the Profiler report. The Profiler generates an AWR report for the interval covered by the snapshots using the package DBMS_WORKLOAD_REPOSITORY. If the Profiler parameter session is set to indicate a valid database session, then an ASH report is also generated. AWR and ASH reports are placed in the same directory as Profiler reports. Both AWR and ASH reports are generated in HTML format.

```
Begin snapshot: 2495
End snapshot: 2496
AWR report: awr_report_2495_2496.html
ASH report: ash_report_2495_2496.html
```

Hardware

This section provides information on the CPU resources that are available to the Oracle DBMS instance that generated the trace file. The number of CPU cores indicated is determined from the Oracle initialization parameter cpu_count.

```
cpu_count=2 (number of CPU cores)
```

Initialization Parameters

This section list all parameters with non-default values as well as parameters with default values that are relevant to performance diagnosis and optimization.

Name	Value	Default	Modified
_ash_sample_all	FALSE	FALSE	FALSE
background_dump_dest	C:\PROGRAMME\ORACLE\PRODUCT\ADMIN\TEN\BDUMP	FALSE	FALSE
compatible	10.2.0.1.0	FALSE	FALSE
control_files	C:\ORADATA\TEN\CONTROL01.CTL, C:\ORADATA\TEN\CONTROL02.CTL, C:\ORADATA\TEN\CONTROL03.CTL	FALSE	FALSE
core_dump_dest	C:\PROGRAMME\ORACLE\PRODUCT\ADMIN\TEN\CDUMP	FALSE	FALSE
create_bitmap_area_size	8388608	TRUE	FALSE
db_block_size	8192	FALSE	FALSE
db_cache_size	28M	FALSE	FALSE
db_domain	oradbpro.com	FALSE	FALSE
db_file_multiblock_read_count	16	FALSE	FALSE
db_name	TEN	FALSE	FALSE
db_16k_cache_size	52M	FALSE	FALSE
db_2k_cache_size	8M	FALSE	FALSE
disk_asynch_io	TRUE	TRUE	FALSE
dispatchers	(protocol=tcp)(dispatchers=1)	FALSE	FALSE
hash_area_size	131072	TRUE	FALSE
java_pool_size	32M	FALSE	FALSE
job_queue_processes	1	FALSE	FALSE
large_pool_size	0	FALSE	FALSE
local_listener	listener.oradbpro.com	FALSE	FALSE
log_archive_dest_1	location=c:\temp	FALSE	FALSE
log_buffer	6984704	TRUE	FALSE
max_dump_file_size	UNLIMITED	TRUE	FALSE
open_cursors	300	FALSE	FALSE
optimizer_dynamic_sampling	2	TRUE	FALSE
optimizer_features_enable	10.2.0.3	TRUE	FALSE
optimizer_index_caching	0	TRUE	FALSE
optimizer_index_cost_adj	100	TRUE	FALSE
optimizer_mode	ALL_ROWS	TRUE	FALSE
optimizer_secure_view_merging	TRUE	TRUE	FALSE
parallel_execution_message_size	2148	TRUE	FALSE
pga_aggregate_target	256M	FALSE	FALSE
processes	50	FALSE	FALSE
resource_limit	TRUE	FALSE	FALSE
resource_manager_plan	SYSTEM_PLAN	FALSE	FALSE
service_names	TEN.oradbpro.com	FALSE	FALSE
shared_pool_reserved_size	10M	TRUE	FALSE
shared_pool_size	200M	FALSE	FALSE
shared_servers	0	FALSE	FALSE
sort_area_size	65536	TRUE	FALSE
statistics_level	TYPICAL	FALSE	FALSE
streams_pool_size	20M	FALSE	FALSE
undo_management	AUTO	FALSE	FALSE
undo_tablespace	UNDOTBS1	FALSE	FALSE
user_dump_dest	C:\PROGRAMME\ORACLE\PRODUCT\ADMIN\TEN\UDUMP	FALSE	FALSE

System Statistics

This section shows the operating system statistics that affect the optimization of SQL statements by the cost based optimizer (CBO). The documented interface package DBMS_STATS is used to retrieve the system statistics. By default only so called “no workload” statistics exist. Workload statistics may be gathered by using the package DBMS_STATS. These provide the optimizer with better information on the hardware capabilities than no workload statistics. Jonathan Lewis, the author of “Cost-Based Oracle Fundamentals” recommends gathering and setting workload statistics. Note that gathering workload statistics will yield varying results over time. It is recommended to gather workload statistics in a statistics table created using DBMS_STATS.CREATE_STAT_TABLE instead of immediatly importing them into the data dictionary base table SYS.AUX_STATS$. Workload statistics from several periods should by scrutinized and sensible values should be set using DBMS_STATS.SET_SYSTEM_STATS.

```
Gathered between 2008-10-21 20:50:00.0 and 2008-10-21 20:50:00.0

Parameter              Value Description
_________ __________________ ____________________________________________________________
cpuspeednw            1386.2 Noworkload CPU speed (million operations/s)
ioseektim               10.0 I/O seek time (ms)
iotfrspeed            4096.0 I/O transfer speed (bytes/ms)
cpuspeed           undefined Workload CPU speed (million operations/s)
maxthr             undefined Maximum I/O system throughput (bytes/s)
mbrc               undefined Average mutli block read count
mreadtim           undefined Multiblock read time (ms/block)
slavethr           undefined Maximum throughput of a parallel execution slave (bytes/s)
sreadtim           undefined Single block read time (ms)
```

DBMS_STATS Default Values

This section lists the default values that are used by the package DBMS_STATS when calculating object statistics for tables and indexes. The default setting DBMS_STATS.AUTO_SAMPLE_SIZE for the DBMS_STATS parameter may cause objects statistics with very low quality, since the fraction of blocks that are sampled may be too low. It is advisable to set estimate_percent to at least 25. Generally it is preferable to have higher quality statistics that are updated on a weekly basis with a higher value for estimate_percent than to have low quality statistics that are upated every day based on the automatic statistics gathering introduced with Oracle10g. It is feasible to gather statistics on a daily basis with the DBMS_STATS parameter “options” set to “GATHER EMPTY” to make sure that all objects have statistics. The higher overhead of using an estimate_percent of 25 may be offset by calculating statistics with “options” set to “GATHER STALE” on a weekly basis.

Poor execution plans may result from inaccurate statistics. If function-based indexes are used, make sure that the hidden columns that implement them have accurate statistics. This may require using a “method_opt” setting of “FOR ALL HIDDEN COLUMNS”. Refer to a MERITS Profiler report in real-time mode or the data dictionary view DBA_TAB_COLS for information on hidden columns pertaining to function-based indexes.

Following is an excerpt from a MERITS Profiler report in text mode that shows DBMS_STATS settings that all have default values.

```
DBMS_STATS Default Values
~~~~~~~~~~~~~~~~~~~~~~~~~

AUTOSTATS_TARGET CASCADE                  DEGREE ESTIMATE_PERCENT             GRANULARITY METHOD_OPT                 NO_INVALIDATE
________________ ________________________ ______ ____________________________ ___________ __________________________ ___________________________
AUTO             DBMS_STATS.AUTO_CASCADE  NULL   DBMS_STATS.AUTO_SAMPLE_SIZE  AUTO        FOR ALL COLUMNS SIZE AUTO  DBMS_STATS.AUTO_INVALIDATE
```

Correlation with V$SQL and V$SQL_PLAN_STATISTICS_ALL

This section provides a lot of valuable data on SQL statement execution that are not available at all in SQL trace files or that may be missing under certain circumstances. For example, if a database client running against Oracle10g is traced, the trace file will only contain execution plans for cursors that were closed while tracing was active. This may result in SQL trace files that do not contain execution plans for several statements. The MERITS Profiler compensates for this by looking up the execution plans for all statements with known hash values and including them in the report. Multiple plans for a single statement are supported. Hence it will be evident whether the plan for a statement has changed. Each distinct execution plan has a separate child number in V$SQL. If multiple plan hash values for a single statement exist, then this indicates that different execution plans have been used. If the Oracle instance runs with statistics_level=ALL additional information on the last execution including actual execution time and buffer gets may be available. The MERITS Profiler is able to set statistics_level=ALL during the measurement interval. Use the Profiler parameter statistics_level (same parameter name and permissible values as with the Oracle DBMS) for this purpose.

Execution plans are retrieved along with additional information such as query block names, peeked bind variables, outline data, and predicate information using the package DBMS_XPLAN. Bind variable peeking is a feature of the CBO that is enabled by default. Outline data contain hints that force a particular execution plan. The hints found in the outline data section are a good starting point for modifying a plan with different hints.

An execution plan may be the result of a SQL Profile are a Stored Outline. This information is not included in SQL trace files. However the Profiler report provides this information.

```
Correlation with V$SQL
----------------------


Note: The result is split into two rows where each row is identified by the child cursor number.
Resource usage is since instance startup (not interval-based).

CHILD_NUMBER OLD_HASH_VALUE FORCE_MATCHING_SIGNATURE PLAN_HASH_VALUE OPTIMIZER_ENV_HASH_VALUE LAST_ACTIVE_TIME   SQL_PROFILE OUTLINE_CATEGORY OUTLINE_TYPE
____________ ______________ ________________________ _______________ ________________________ __________________ ___________ ________________ ____________
           0     3803592478      3138049466602010507      2108007948               3522870812 06-Jan-10 18:39:21

CHILD_NUMBER CPU Time (s) Elapsed (s) App. Wait Time (s) Conc. Wait Time (s) Cluster Wait Time (s) User I/O Wait Time (s) PL/SQL Exec. Time (s) Java Exec. Time (s) EXECUTIONS SORTS PARSE_CALLS OPTIMIZER_COST
____________ ____________ ___________ __________________ ___________________ _____________________ ______________________ _____________________ ___________________ __________ _____ ___________ ______________
           0 15.825       54.881      0                  3.715               0                     28.803                 0                     0                         2066  2066          15            235

Correlation with V$SQL_PLAN_STATISTICS_ALL using DBMS_XPLAN.DISPLAY_CURSOR
--------------------------------------------------------------------------


Execution Plan for Child Cursor 0
.................................

SQL_ID  dw2zgaapax1sg, child number 0
-------------------------------------
SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME, PRODUCT_DESCRIPTION, CATEGORY_ID, WEIGHT_CLASS, WARRANTY_PERIOD, SUPPLIER_ID,
PRODUCT_STATUS, LIST_PRICE, MIN_PRICE, CATALOG_URL, QUANTITY_ON_HAND FROM PRODUCTS, INVENTORIES WHERE PRODUCTS.CATEGORY_ID = :B1
AND INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID ORDER BY INVENTORIES.WAREHOUSE_ID

Plan hash value: 2108007948

-----------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name                  | E-Rows |E-Bytes|E-Temp | Cost (%CPU)| E-Time   |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------
|   1 |  SORT ORDER BY                  |                       |    580 |   288K|   632K|   235   (1)| 00:00:03 |   267K|   267K|  237K (0)|
|*  2 |   HASH JOIN                     |                       |    580 |   288K|       |   169   (1)| 00:00:03 |   694K|   694K| 1156K (0)|
|   3 |    NESTED LOOPS OUTER           |                       |     29 | 14500 |       |     4   (0)| 00:00:01 |       |       |          |
|*  4 |     TABLE ACCESS FULL           | PRODUCT_INFORMATION   |     29 |  6351 |       |     4   (0)| 00:00:01 |       |       |          |
|   5 |     TABLE ACCESS BY INDEX ROWID| PRODUCT_DESCRIPTIONS |      1 |   281 |       |     0   (0)|          |       |       |          |
|*  6 |      INDEX UNIQUE SCAN          | PRD_DESC_PK           |      1 |       |       |     0   (0)|          |       |       |          |
```

```
|   7 |    TABLE ACCESS FULL           | INVENTORIES         |   5760 | 57600 |       |   164   (0)| 00:00:02 |       |       |         |
-------------------------------------------------------------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$F5BB74E1
   4 - SEL$F5BB74E1 / I@SEL$2
   5 - SEL$F5BB74E1 / D@SEL$2
   6 - SEL$F5BB74E1 / D@SEL$2
   7 - SEL$F5BB74E1 / INVENTORIES@SEL$1

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('10.2.0.3')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$F5BB74E1")
      MERGE(@"SEL$2")
      OUTLINE(@"SEL$1")
      OUTLINE(@"SEL$2")
      FULL(@"SEL$F5BB74E1" "I"@"SEL$2")
      INDEX_RS_ASC(@"SEL$F5BB74E1" "D"@"SEL$2" ("PRODUCT_DESCRIPTIONS"."PRODUCT_ID" "PRODUCT_DESCRIPTIONS"."LANGUAGE_ID"))
      FULL(@"SEL$F5BB74E1" "INVENTORIES"@"SEL$1")
      LEADING(@"SEL$F5BB74E1" "I"@"SEL$2" "D"@"SEL$2" "INVENTORIES"@"SEL$1")
      USE_NL(@"SEL$F5BB74E1" "D"@"SEL$2")
      USE_HASH(@"SEL$F5BB74E1" "INVENTORIES"@"SEL$1")
      END_OUTLINE_DATA
  */

Peeked Binds (identified by position):
--------------------------------------

   1 - (NUMBER): 9

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("INVENTORIES"."PRODUCT_ID"="I"."PRODUCT_ID")
   4 - filter("I"."CATEGORY_ID"=:B1)
   6 - access("D"."PRODUCT_ID"="I"."PRODUCT_ID" AND "D"."LANGUAGE_ID"=SYS_CONTEXT('USERENV','LANG'))

Note
-----
   - Warning: basic plan statistics not available. These are only collected when:
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level
```

Execution Plans Captured by Statspack

If the Statspack repository contains one or more execution plans for a traced statement, then these are included in the Profiler report given that the Profiler parameter use_statspack has the value true. In case additional information on a statement from the Statspack repository is desired, it can be obtained by using the Statspack script sprepsql.sql. The old hash value and the Statspack snapshot ID indicated in the report are required as input to the script sprepsql.sql.

```
Old hash value: 3803592478
SQL ID: dw2zgaapax1sg

Execution plan with plan hash value 2108007948 (last active 15-Nov-09 19:47:59)
Optimization: ALL_ROWS
Cost: 235
Statspack snapshot 505 of instance 1

ID PID Operation                      Object                   Object ID Rows  Bytes Cost I/O Cost Temp. Space
__ ___ _____________________________ ________________________ _________ ____ ______ ____ ________ ____________
 0     SELECT STATEMENT                                                    235
 1   0  SORT ORDER BY                                             580 295800  235      233       648000
 2   1   HASH JOIN                                                580 295800  169      168
 3   2    NESTED LOOPS OUTER                                       29  14500    4        4
 4   3     TABLE ACCESS FULL          SOE.PRODUCT_INFORMATION    60741   29   6351    4        4
 5   3     TABLE ACCESS BY INDEX ROWID SOE.PRODUCT_DESCRIPTIONS  60743    1    281    0        0
 6   5      INDEX UNIQUE SCAN         SOE.PRD_DESC_PK            60770    1           0        0
 7   2    TABLE ACCESS FULL           SOE.INVENTORIES            60740 5760  57600  164      164
```

Statement Execution Captured by AWR

If the Active Workload Repository contains one or more execution plans for a traced statement, then these are included in the Profiler report given that the Profiler parameter use_awr has the value true. In case additional information on a statement from the AWR is desired, it can be obtained by using the AWR script awrsqrpt.sql. The begin snapshot ID and the SQL ID are required as input to the script awrsqrpt.sql.

```
SNAP_ID      DBID INSTANCE_NUMBER PLAN_HASH_VALUE OPTIMIZER_COST OPTIMIZER_MODE OPTIMIZER_ENV_HASH_VALUE SQL_PROFILE Ela. per EXEC CPU per EXEC Buffer Gets per EXEC
_______ _________ _____________ _______________ _____________ _____________ _____________________ ___________ _____________ _____________ _____________________
   2492 2870266532             1      2108007948           235 ALL_ROWS                 3522870812           .019332       .007719      742.2


AWR execution plan with plan hash value 2108007948 for SQL ID dw2zgaapax1sg
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


SQL_ID dw2zgaapax1sg
--------------------
SELECT PRODUCTS.PRODUCT_ID, PRODUCT_NAME, PRODUCT_DESCRIPTION, CATEGORY_ID, WEIGHT_CLASS,
```

```
WARRANTY_PERIOD, SUPPLIER_ID, PRODUCT_STATUS, LIST_PRICE, MIN_PRICE, CATALOG_URL, QUANTITY_ON_HAND FROM
PRODUCTS, INVENTORIES WHERE PRODUCTS.CATEGORY_ID = :B1 AND INVENTORIES.PRODUCT_ID = PRODUCTS.PRODUCT_ID
ORDER BY INVENTORIES.WAREHOUSE_ID

Plan hash value: 2108007948

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                | E-Rows |E-Bytes|E-Temp | Cost (%CPU)| E-Time   |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                     |        |       |       |   235 (100)|          |
|   1 |  SORT ORDER BY                |                     |    580 |   288K|   632K|   235   (1)| 00:00:03 |
|   2 |   HASH JOIN                   |                     |    580 |   288K|       |   169   (1)| 00:00:03 |
|   3 |    NESTED LOOPS OUTER         |                     |     29 | 14500 |       |     4   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL         | PRODUCT_INFORMATION |     29 |  6351 |       |     4   (0)| 00:00:01 |
|   5 |     TABLE ACCESS BY INDEX ROWID| PRODUCT_DESCRIPTIONS |      1 |   281 |       |     0   (0)|          |
|   6 |      INDEX UNIQUE SCAN        | PRD_DESC_PK         |      1 |       |       |     0   (0)|          |
|   7 |    TABLE ACCESS FULL          | INVENTORIES         |   5760 | 57600 |       |   164   (0)| 00:00:02 |
-----------------------------------------------------------------------------------------------------------

Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------

   1 - SEL$F5BB74E1
   4 - SEL$F5BB74E1 / I@SEL$2
   5 - SEL$F5BB74E1 / D@SEL$2
   6 - SEL$F5BB74E1 / D@SEL$2
   7 - SEL$F5BB74E1 / INVENTORIES@SEL$1

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('10.2.0.3')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$F5BB74E1")
      MERGE(@"SEL$2")
      OUTLINE(@"SEL$1")
      OUTLINE(@"SEL$2")
      FULL(@"SEL$F5BB74E1" "I"@"SEL$2")
      INDEX_RS_ASC(@"SEL$F5BB74E1" "D"@"SEL$2" ("PRODUCT_DESCRIPTIONS"."PRODUCT_ID"
            "PRODUCT_DESCRIPTIONS"."LANGUAGE_ID"))
      FULL(@"SEL$F5BB74E1" "INVENTORIES"@"SEL$1")
      LEADING(@"SEL$F5BB74E1" "I"@"SEL$2" "D"@"SEL$2" "INVENTORIES"@"SEL$1")
      USE_NL(@"SEL$F5BB74E1" "D"@"SEL$2")
      USE_HASH(@"SEL$F5BB74E1" "INVENTORIES"@"SEL$1")
      END_OUTLINE_DATA
  */

Peeked Binds (identified by position):
--------------------------------------

   1 - :B1 (NUMBER): 1

Note
-----
   - Warning: basic plan statistics not available. These are only collected when:
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level
```

Optimizer Environments

An optimizer environment is a set of CBO parameters with certain values. An optimizer environment hash value is calculated for each optimizer environment. The optimizer environment hash value offers an easy way to tell whether different optimizer environments were used by the CBO when optimizing a set of statements. Since optimizer parameters may be changed with the command ALTER SESSION it is feasible for an application to change optimizer parameters. Parameters may also have changed with ALTER SYSTEM. Changes with ALTER SYSTEM are picked up by sessions that connect after the parameter change.

```
This section contains cost based optimizer (CBO) parameter settings pertaining to
execution plans used by the SQL statements in the trace file. An optimizer environment
is a distinct set of parameters that govern the operation of the optimizer.
An optimizer environment hash value (V$SQL.OPTIMIZER_ENV_HASH_VALUE) identifies each unique set
of parameter values.

The optimizer used 4 distinct optimizer environments.

Optimizer Environment with Hash Value 779777996
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


This optimizer environment was used when generating an execution plan for the cursor
identified by SQL ID bsa0wjtftg3uw and child number 0.
```

Row#	Name	Value	Default
1	active_instance_count	1	YES
2	bitmap_merge_area_size	1048576	YES
3	cpu_count	2	YES
4	cursor_sharing	exact	YES
5	hash_area_size	131072	YES
6	optimizer_dynamic_sampling	2	YES
7	optimizer_features_enable	10.2.0.3	YES
8	optimizer_index_caching	0	YES
9	optimizer_index_cost_adj	100	YES
10	optimizer_mode	choose	NO
11	optimizer_secure_view_merging	true	YES
12	parallel_ddl_mode	enabled	YES
13	parallel_dml_mode	disabled	YES
14	parallel_execution_enabled	true	YES
15	parallel_query_mode	enabled	YES
16	parallel_threads_per_cpu	2	YES
17	pga_aggregate_target	262144 KB	YES
18	query_rewrite_enabled	true	YES
19	query_rewrite_integrity	enforced	YES
20	skip_unusable_indexes	true	YES
21	sort_area_retained_size	0	YES
22	sort_area_size	65536	YES
23	star_transformation_enabled	false	YES
24	statistics_level	typical	YES
25	workarea_size_policy	auto	YES

Buffer Cache Contents

This section shows the contents of the buffer cache. Segments that are smaller than the value of the MERITS profiler parameter cached_table_threshold_mb are exempt from the report. The overall load on the disk subsystem may be reduced significantly if separate buffer pools (KEEP, RECYCLE, DB_*n*K_CACHE_SIZE) are used for large tables or indexes that cause many physical reads. Consult the Statspack level 7 report section on 'Segments by Physical Reads' to identify segments that cause many physical reads and are hence insufficiently cached.

Owner & Name	Object Type	Block Size (KB)	Buffer Pool	Cached (MB)	Cached (%)	Cached (Blocks)	Segment Size (MB)	Segment Size (Blocks)
SYS.C_OBJ#	CLUSTER	8	DEFAULT	7.1	88.96	911	8	1024
SYS.OBJ$	TABLE	8	DEFAULT	5	83.98	645	6	768
PERFSTAT.STATS$SQL_PLAN_USAGE	TABLE	8	DEFAULT	3.9	78.59	503	5	640
SYS.C_FILE#_BLOCK#	CLUSTER	8	DEFAULT	1.2	59.77	153	2	256

Data Dictionary Correlation

This section contains structural information and cost based optimizer (CBO) statistics pertaining to the tables and indexes referenced by an input trace file. The data is extracted from dictionary views such as DBA_TABLES, DBA_TAB_COLUMNS and DBA_INDEXES.

Structure, Indexes, and Statistics for Table HR.DEPARTMENTS

Table Overview

Owner & Name	Avg. Row Length	Rows (Statistics)	Rows (Est. Actual)	Blocks (Stats)	Blocks (Segment)	Empty Blocks	Average Space	Chain Count	Global Stats	User Stats	Sample Size	Last Analyze
HR . DEPARTMENTS	20	27	38	5	8	0	0	0	YES	NO	27	21-May-09 18:50:15

Table Size and Estimated Space Efficiency

Tablespace	Block Size	Table Size (MB)	Est. Space Efficiency	Buffer Pool	Degree	Cluster	IOT Type	IOT Name
USERS	8 KB	.1	1.5%	DEFAULT	1			

Storage Parameters and Modifications

PCTFREE	PCTUSED	INITRANS	Monitoring	Modified (%)	INSERT	UPDATE	DELETE
10		1	YES				

Column Statistics

Column Name	Data Type	Distinct Values	Avg. Length	Density	Buckets	NULLs	Global Stats	User Stats	Sample Size	Last Analyze
DEPARTMENT_ID	NUMBER(4) NOT NULL	27	4	0.037037	1	0	YES	NO	27	21-May-09 18:50:15
DEPARTMENT_NAME	VARCHAR2(30) NOT NULL	27	12	0.037037	1	0	YES	NO	27	21-May-09 18:50:15
MANAGER_ID	NUMBER(6)	11	3	0.090909	1	16	YES	NO	11	21-May-09 18:50:15
LOCATION_ID	NUMBER(4)	7	3	0.018519	7	0	YES	NO	27	21-May-09 18:50:15

Index Overview

```
--------------

Index Owner Index Name       Index Type Partitioned Created            Last DDL Time      Last Analyze       Status Sample Size Tablespace Buffer Pool Degree
___________ _______________ _____________ ___________ ___________________ ___________________ ___________________ _____ ___________ ___________ ___________ ______
HR          DEPT_ID_PK       NORMAL     NO          29-Jan-09 00:40:34 29-Jan-09 00:40:34 21-May-09 18:50:15 VALID           27 USERS      DEFAULT     1
HR          DEPT_LOCATION_IX NORMAL     NO          29-Jan-09 00:40:35 29-Jan-09 00:40:35 21-May-09 18:50:15 VALID           27 USERS      DEFAULT     1

Index Statistics
................

Index Owner Index Name       Index Type Unique B-Tree Level Leaf Blocks Distinct Keys Rows Avg. Leaf Blocks per Key Avg. Data Blocks per Key Clustering
Factor Global Stats User Stats
___________ _______________ _____________ _____ _____________ ___________ _____________ ____ ___________________________ ___________________________
_____________ ___________ ___________
HR          DEPT_ID_PK       NORMAL     YES               0           1            27   27                         1                         1                1
YES           NO
HR          DEPT_LOCATION_IX NORMAL     NO                0           1             7   27                         1                         1                1
YES           NO

Indexed Columns
...............

Index Owner Index Name       Column Name   Position Data Type
___________ _______________ _____________ ________ ________________
HR          DEPT_ID_PK       DEPARTMENT_ID        1 NUMBER(4) NOT NULL
HR          DEPT_LOCATION_IX LOCATION_ID          1 NUMBER(4)
```

MERITS Profiler Parameter Settings

The final section of the Profiler report documents the parameter settings that were in effect for the report. The next few lines show an excerpt from a section on parameter settings.

```
MERITS Profiler Parameter Settings
__________________________________


awr_flush_level=off
cached_table_threshold_mb=1
date_format=%1$td-%1$tb-%1$ty %1$tT
db_directory=
db_release=
```

Logging

The MERITS Profiler's logging subsystem is based on Apache log4j (http://logging.apache.org). Log4j supports seven logging levels. They are TRACE, DEBUG, INFO, WARN, ERROR, FATAL, and OFF in diminishing order of verbosity.

The Profiler consists of five major building blocks. The profiler consists of components responsible for accounting, calculation (calc), parsing (parser), near real-time access to the Oracle DBMS (rt), and the profiler (profiler) itself. The logging level may be configured individually for each building block. The default logging levels are defined in the properties file <MPROF_HOME>/conf/profiler_log4j.properties. To customize logging levels, copy the file profiler_log4j.properties to a directory outside of <MPROF_HOME> and edit the last five lines (reproduced below) to suit your needs.

```
log4j.logger.accounting=INFO
log4j.logger.calc=INFO
log4j.logger.parser=INFO
log4j.logger.profiler=INFO
log4j.logger.rt=INFO
```

Additionally you need to set the Profiler parameter log4j_config_url (see page 24) such that the Profiler will read the customized file instead of the default file in <MPROF_HOME>/conf.

The parser has an interesting feature that is useful for troubleshooting situations where you need to know at what time SQL or PL/SQL statements in a trace file were executed. At logging level DEBUG, the parser writes each trace file entry that was parsed successfully to the log file using a "normalized" format. The format consists of the line number of the entry in a trace file, the character "+" that signifies a successful parse operation, the entry itself including cursor number and hash value, and a human readable timestamp for all trace file lines that contain the parameter tim. The tim value, which has microsecond resolution, is converted to a human readable timestamp with millisecond resolution based on the previous date and time information recorded in the trace file.

The next code section shows an example:

```
DEBUG parser: 36+PARSING IN CURSOR #1 hv=2228079888 dep=0 len=87 uid=61 oct=2 lid=61
ad='6cad992c' sqlid=null tim=789991639097 20-Nov-07 15:39:38.552
DEBUG parser: 39+ INSERT INTO customer(name, phone) VALUES (:name, :phone)
        RETURNING id INTO :id

DEBUG parser: 40+PARSE #1 hv=2228079888 dep=0 c=0 e=84 p=0 cr=0 cu=0 mis=0 r=0 og=1 plh=null
tim=789991639091 20-Nov-07 15:39:38.552
```

Note that the parameter hv is present in the normalized parse call from the log file excerpt above. SQL trace files provide hv only with PARSING IN CURSOR entries. The human readable timestamp "20-Nov-07 15:39:38.552" is derived from the value of the tim parameter on the same line.

A minus sign is used to signify trace file entries that failed to parse without error. Here is an example:

```
ERROR parser: 237613- =0  =0 obj#=0 tim=113720081170
```

Here, the integer 237613 is the line number in a trace file. The minus sign after the line number indicates a parse error. The error above occurred due to a bug in the DBMS server that is the cause for WAIT entries with missing wait parameter names (“=0” is written instead of <wait parameter name>=0).

Releases and Features

New Features of Release 0.9.11

Release 0.9.11 has the following new features:

- Parsing of row sources in STAT entries has been greatly enhances such that new row sources that are not explicitly supported by the lexer no longer cause a failure to parse the metrics at the end of a STAT entry. Note that each release of the Oracle DBMS may introduce new row sources and/or options such as “HASH JOIN FULL OUTER”.
- Microsecond resolution for “Avg. SBR Time” and “Avg. MBR Time” (average single block read time and average multiblock read time).
- Support for ERROR entries and a counter that is used to report the number of ERROR entries (Oracle DBMS server ORA-nnnnn errors) in a trace file.

New Features of Release 1.0.0

Release 1.0.0 supports Oracle Version 11.2.0.2 SQL trace files with 64-bit unsigned integers as cursor numbers. Previous releases do not support 64-bit unsigned integers as cursor numbers. Release 1.0.0 is also the first release that limits the size of trace files processed. The maximum trace file size is encoded in a license file. License files issued for previous versions of the Profiler are not compatible with release 1.0.0.

The value of the parameter think_time_threshold_ms has been reduced from 5 to 4 since today’s networks easily allow for round-trip times of less than 4 ms. Furthermore 4 ms is a histogram bucket boundary such that the wait event histogram for “SQL*Net message from client” is easier to interpret.

New features:

- Accounting by module and action
- LOB operation accounting
- Support for CLOSE database call (parser and accounting)

New Features of Release 1.1.0

The parser now supports the row source INDEX FULL SCAN (MIN/MAX) in execution plans captured by SQL trace.

New Features of Release 1.2.0

- The columns HIDDEN_COLUMN and VIRTUAL_COLUMN from the view DBA_TAB_COLS have been added to the section entitled “Column Statistics” to better support function based indexes.
- The section “DBMS_STATS Default Values” has been added to MERITS Profiler reports in real-time mode. The default values used by DBMS_STATS affect the quality of segment statistics which in turn have a very significant impact upon execution plans chosen by the cost based optimizer (CBO).
- The initialization parameters db_file_multiblock_read_count and instance_name are now included in the section entitled “Initialization Parameters” even if they have default values. The parameter db_file_multiblock_read_count affects the cost based optimizer (CBO). Values higher than 16 rarely have a measureable benefit. Nonetheless in Oracle10*g* Oracle Corp. increased the default to 128 on many platforms given that db_block_size=8192. The default value of the parameter db_file_multiblock_read_count is set such that db_file_multiblock_read_count * db_block_size = 10485876 (1 MB).
- Modules and actions are now ranked by delta tim spent inside a module and action in the report section “Results by Module and Action”. Earlier releases used the accounted for elapsed time to rank modules and actions. Using delta tim provides more accurate results when the Oracle DBMS kernel performs operations that aren’t sufficiently instrumented.
- The row source operation "DOMAIN INDEX" now parses successfully.
- The MERITS Profiler parameter setting cached_table_threshold_mb=-1 disables the real-time report section entitled "Buffer Cache Contents".

Bug Fixes in Release 1.2.0

- Trace files that contained a row source with obj# > 2^{31}-1 caused an integer overflow. This could happen with GV$ or V$ views.
- Values larger than 2^{31}-1 in the row source metric fields card or size caused an integer overflow.
- The release information in Standard Edition trace files could not be extracted.

- LOB and LOB index segments were missing in the section entitled "Buffer Cache Contents". To implement the fix, a GRANT on the dictionary base table OBJ$ has been added in the file profiler_role.sql. If you use the role created by this script, then you need to execute the script again to make sure the new release does not cause ORA-00942: table or view does not exist.

Upgrading

This section contains instructions for upgrades. If your old release is older than a certain newer release, then you need to follow the instructions for all releases higher than your old release up to and including the release you are upgrading to. For example if your old release is 0.9.3 and your new release is 0.9.5, you need to follow the instructions for release 0.9.5 (there are no upgrade tasks for release 0.9.4).

Upgrading to Release 0.9.5

The startup scripts (mprof/mprof.bat) in the directory mprof/bin have changed. In prior releases the startup scripts themselves had to be modified to launch the Profiler. Starting with release 0.9.5 the startup scripts expect the environment variables MPROF_HOME and MPROF_JAVA_HOME to be set before a startup script can be invoked successfully. It is recommended to copy the customized settings of MPROF_HOME and JAVA_HOME from an older release and to set the variables MPROF_HOME and MPROF_JAVA_HOME as environment variables using the already established values.

Upgrading to Release 1.0.0

To upgrade an existing MPROF_HOME simply overwrite all files using unzip -o or a similar approach. Release 1.0.0 is packaged as zip file mprof-1.0.0.zip.

Upgrading to Release 1.1.0

To upgrade an existing MPROF_HOME simply overwrite all files using unzip -o or a similar approach. Release 1.1.0 is packaged as zip file mprof-1.1.0.zip.

Upgrading to Release 1.2.0

A GRANT on the view DBA_TAB_COLS has been added in the file profiler_role.sql. If you use the role created by this script, then you need to execute the script again to make sure the new release does not cause ORA-00942: table or view does not exist while trying to access the view DBA_TAB_COLS.

Limitations

This section describes some limitations of the MERITS Profiler.

Partitioning Option

Partitioned tables and indexes are not supported in real-time mode. Support for Partitioning Option is under development.

Multiple Sessions per Trace File

The MERITS Profiler currently has no support for correctly handling cursors from multiple sessions in a single file. This applies to trace files from shared server processes as well as concatenated trace files created manually or using TRCSESS. Such files may be processed but results will likely be inaccurate. As a workaround, use TRCSESS to create one or more trace files that contain only a single session, then process those trace files with the MERITS Profiler.

This limitation is scheduled to be removed within the first quarter of 2010.

Parallel Execution

The MERITS Profiler cannot detect parallel execution of a SQL statement. There is currently no feature for accurately combining the SQL trace file of the parallel execution coordinator process and two or more trace files of parallel execution slaves controlled by the coordinator into a single report.

LOB Statsistics at Module and Action Level

Accounting by module and action ignores LOB trace file entries. LOB statistics are only available at trace file level, not at module and action level.

Additional Information

Please consider the following books for additional information on Oracle DBMS performance optimization.

Antognini, Christian, *Troubleshooting Oracle Performance*, Apress, 2008, http://www.apress.com

Debes, Norbert, *Secrets of the Oracle Database*, Apress, 2009, http://www.apress.com

Lewis, Jonathan, *Cost–Based Oracle Fundamentals*, Apress, 2005, http://www.apress.com

Millsap, Cary; Holt, Jeff, *Optimizing Oracle Performance*, O'Reilly, 2003, http://www.oreilly.com

Index

U

V

W

X