Addressing a performance issue: the drilldown approach

Laurent Leturgez
PREMISEO
Lille - FRANCE

Keywords:

ASH, AWR, Statspack, DB Time, Database load, Active wait time, Idle wait time, DB CPU, Active Session History, dataviz, heatmap, code instrumentation, analysis, PL/SQL, SQL, Java, Profiling

Introduction

Oracle performance issues are rarely fully qualified by end-users.
Usually, they come to the DBA with this fuzzy message "my application is slow, can you help me ?".
So the DBAs have to determine when and how it is slowed down and to do that, they have to use the good methodology.

During this session, a method will be described to identify problems from the general (It's slow!!) to the detail, for example, a PLSQL issue.
To do this, two distinct parts will be shown in order to diagnose bottlenecks (and only diagnose):
- First part, a technique that will help to identify hot points will be detailed. Once identified, it will be easier to point out when the problem occurs and then focus the analyse on this time period.
This technique will help DBAs to guide their investigation on a CPU time reduction or on a wait time reduction.
The presentation of this topic will be done with and without the help of Diagnostic pack.
- Second part, an overall view of available tools will be done and how they can be used to detect bottlenecks, top sql etc.
AWR/Statspack, ASH, and the PLSQL profiler are very powerful tools, and for each tool, one example from real cases will be presented.

Finally, with these methods and tools, the attendees will be able to easily identify bottlenecks from the general to the detail, and then apply the fixes that will correct the performance issues.

It’s always a question of time

Addressing a performance issue always starts by a diagnostic phase. In this approach, analysing how time is consumed by the database is a key point.

Time analysis requires to know how an Oracle session can spend time:

- First, a session can wait for work to do. In this case, the session accounts Idle Wait Time.
- If the session waits for a system call or something to complete (it can be an I/O, or the session can wait on a lock etc.). In this case, the session waits on an active wait time event.
- If the session executes Oracle code on CPU, it will account DB CPU Time.

The most important time to measure is the database time (DB Time). DB Time is the sum of Active Wait Time and DB CPU Time.

Active average session and database load

Another interesting ratio is Active Average Session (AAS). AAS is equal to DB Time divided by Elapsed Time.
AAS represents the average number of sessions that are working or waiting actively in the database.
When AAS becomes greater than the number of CPU Core, it can be interesting to analyse this period of time.

Finally, from AAS we can calculate the database load.

DB Time, AAS and database load are the key values if you want to know if your database is overloaded or not.

Identify bottleneck with various tools

All the previous key values are important. Of course, if your database is overloaded all the time, the problem is a serious one but you know that the issue is general to the database.
The reality is a bit different because usually, performance problems occur periodically or on specific periods of time.

That's why, it's better when you can visualize how time is consumed, and how the database is loaded over the time.

To do that, a time analysis can be done by querying AWR views, ASH views and Statspack schema. Once done, you can use your own tools to visualize this and very easily point out bottlenecks periods.

In the figures below, two kinds of data visualization are shown: a heatmap based on AWR and ASH data and DB Time/DB CPU trending from statspack data.

MTIME	00-01_	01-02_	02-03_	03-04_	04-05_	05-06_	06-07_	07-08_	08-09_	09-10_	10-11_	11-12_	12-13_	13-14_	14-15_	15-16_	16-17_	17-18_	18-19_	19-20_	20-21_	21-22_	22-23_	23-24_
27/03/2017	0	0	0	0	0	0	0	0	0	0	0	0	0,93	1,3	2,33	2,51	2,47	2,39	0,72	0,32	0,21	0,26	0,62	0,91
28/03/2017	0,69	0,47	0,58	0,19	0,15	0,15	0,18	0,19	0,43	1,21	2,3	2,79	1,26	1,02	2,25	2,78	2,1	1,8	1,21	0,42	0,31	0,26	1,16	1,74
29/03/2017	0,39	0,38	0,53	0,24	0,17	0,17	0,24	0,28	0,64	1,54	2,7	2,92	1,58	1,31	1,9	1,77	1,56	1,48	0,88	0,61	0,45	0,44	0,82	1,06
30/03/2017	0,93	0,83	0,75	0,46	0,36	0,36	0,37	0,49	0,62	1,18	1,79	1,6	2,18	1,32	1,49	2,93	1,53	1,96	0,78	0,51	0,39	0,41	1,23	0,93
31/03/2017	0,5	0,44	0,68	0,35	0,34	0,33	0,34	0,45	0,73	1,21	1,38	2,48	1,2	1	1,54	1,53	1,03	1,98	1,28	0,31	0,2	0,19	0,58	0,4
01/04/2017	0,26	0,43	0,52	0,18	0,17	0,16	0,27	0,27	0,28	0,36	0,36	0,37	0,36	0,36	0,28	0,28	0,28	0,26	0,4	0,28	0,29	0,26	0,45	0,35
02/04/2017	0,28	0,28	0,48	0,16	0,17	0,17	0,28	0,28	0,27	0,25	0,3	0,27	0,28	0,27	0,29	0,28	0,32	0,29	0,3	0,3	0,34	0,3	0,47	0,97
03/04/2017	0,97	0,93	0,75	0,2	0,16	0,17	0,58	1,09	0,52	1,68	2,21	3,56	1,78	1,31	2,11	2,46	2,28	1,88	0,6	0,39	0,25	0,4	0,69	0,72
04/04/2017	0,46	0,24	0,52	0,18	0,19	0,14	0,15	0,19	1,39	1,4	2,58	3,23	2,37	1,05	2,38	1,81	2,53	1,8	1,53	0,83	0,17	0,79	0,53	0,28
05/04/2017	0,16	0,3	0,43	0,1	0,08	0,07	0,08	0,12	1,13	1,79	2,19	2,49	1,76	0,62	2,15	2,43	2,37	2,33	1,98	0,93	0,1	0,12	0,46	0,76
06/04/2017	0,43	0,23	0,39	0,23	0,19	0,29	0,27	0,51	0,48	1,98	1,68	2,09	1,26	0,72	1,34	2,34	1,99	1,71	0,45	0,67	0,58	0,25	0,61	0,49
07/04/2017	0,38	0,36	0,56	0,23	0,2	0,19	0,2	0,23	0,9	1,47	1,21	1,77	0,94	0,69	1,32	2,07	1,38	1	0,41	0,4	0,22	0,21	0,57	0,38
08/04/2017	0,32	0,5	0,54	0,25	0,21	0,19	0,3	0,31	0,3	0,41	0,4	0,42	0,39	0,38	0,33	0,35	0,31	0,3	0,32	0,29	0,32	0,29	0,47	0,35
09/04/2017	0,29	0,35	0,57	0,22	0,18	0,18	0,29	0,29	0,29	0,29	0,32	0,3	0,35	0,29	0,3	0,3	0,3	0,29	0,32	0,35	0,33	0,31	0,47	0,95
10/04/2017	0,96	0,95	0,74	0,25	0,19	0,18	0,19	0,23	0,77	1,46	1,13	1,79	1,04	0,75	1,4	1,25	1,28	0,85	0,4	0,25	0,24	0,22	0,61	0,56
11/04/2017	0,29	0,35	0,49	0,26	0,2	0,2	0,2	0,26	0,67	0,81	1,37	1,32	0,81	0,67	1,56	1,46	1,7	1,31	0,52	0,26	0,23	0,27	0,6	0,45
12/04/2017	0,33	0,34	0,44	0,27	0,2	0,19	0,21	0,33	0,51	1,37	1,77	1,79	1,08	0,9	1,62	1,63	1,38	0,94	0,57	0,31	0,23	0,23	0,61	0,76
13/04/2017	0,35	0,41	0,44	0,27	0,19	0,2	0,21	0,26	0,46	1,13	1,38	1,71	0,91	0,98	1,96	1,29	1,51	1,61	1,21	0,3	0,22	0,23	0,69	0,44
14/04/2017	0,29	0,29	0,46	0,24	0,2	0,21	0,21	0,42	0,43	1,26	1,3	1,91	0,83	0,69	1,43	1,07	0,89	0,64	0,4	0,53	0,25	0,33	0,59	0,47
15/04/2017	0,35	0,3	0,45	0,26	0,2	0,19	0,32	0,31	0,32	0,44	0,39	0,44	0,41	0,44	0,28	0,24	0,25	0,36	0,31	0,33	0,32	0,32	0,46	0,3
16/04/2017	0,23	0,21	0,51	0,29	0,21	0,19	0,32	0,36	0,31	0,31	0,31	0,32	0,3	0,31	0,33	0,55	0,33	0,4	0,32	0,32	0,31	0,31	0,48	0,98
17/04/2017	0,98	0,97	0,8	0,26	0,2	0,19	0,2	0,21	0,21	0,22	0,22	0,23	0,24	0,25	0,23	0,24	0,27	0,23	0,23	0,23	0,22	0,3	0,58	0,37
18/04/2017	0,29	0,77	0,48	0,27	0,2	0,2	0,21	0,32	0,63	1,33	1,99	2,1	1,76	1,05	1,57	2,14	2,13	1,64	0,72	0,37	0,28	0,3	0,73	0,82
19/04/2017	0,6	0,32	0,5	0,25	0,21	0,19	0,21	0,34	0,67	1,54	2,34	3,69	1,56	1,39	2,17	2,77	2,49	2,57	0,99	0,56	0,43	0,5	0,75	0,96
20/04/2017	0,66	0,52	0,44	0,28	0,18	0,17	0,18	0,24	0,67	1,44	2,16	2,41	1,89	0,99	2,3	2,48	2,67	2,18	1,06	0,5	0,44	0,54	0,65	0,45
21/04/2017	0,37	0,33	0,43	0,24	0,27	0,19	0,16	0,21	0,7	2,1	2,56	3,38	1,54	1,05	2,04	1,86	1,45	1,6	0,63	0,24	0,15	0,25	0,28	0,34
22/04/2017	0,35	0,34	0,51	0,12	0,13	0,11	0,22	0,22	0,22	0,22	0,23	0,26	0,26	0,22	0,27	0,34	0,24	0,24	0,37	0,24	0,54	0,36	0,22	0,23
23/04/2017	0,16	0,16	0,35	0,15	0,14	0,15	0,25	0,27	0,25	0,25	0,36	0,94	0,27	0,29	0,25	0,3	0,31	0,42	0,37	0,3	0,29	0,43	0,45	0,39
24/04/2017	0,41	0,28	0,44	0,14	0,15	0,14	0,15	0,19	0,96	1,97	2,82	2,99	2,44	1,85	2,91	2,9	2,76	2,47	1,43	0,76	0,44	0,38	0,62	0,47
25/04/2017	0,42	0,44	0,44	0,29	0,33	0,34	0,34	0,4	0,72	1,98	2,49	2,61	1,75	0,7	0,26	2,7	2,9	2,59	1,63	1,52	1,24	0,28	0,44	0,91
26/04/2017	0,48	0,25	0,18	0,15	0,13	0,14	0,14	0,37	0,71	2,23	2,15	2,78	1,25	1,43	2,49	2,44	2,83	2,75	0	0	0	0	0	0

Illustration. 1: Heatmap based on ASH and AWR data to identify peak time

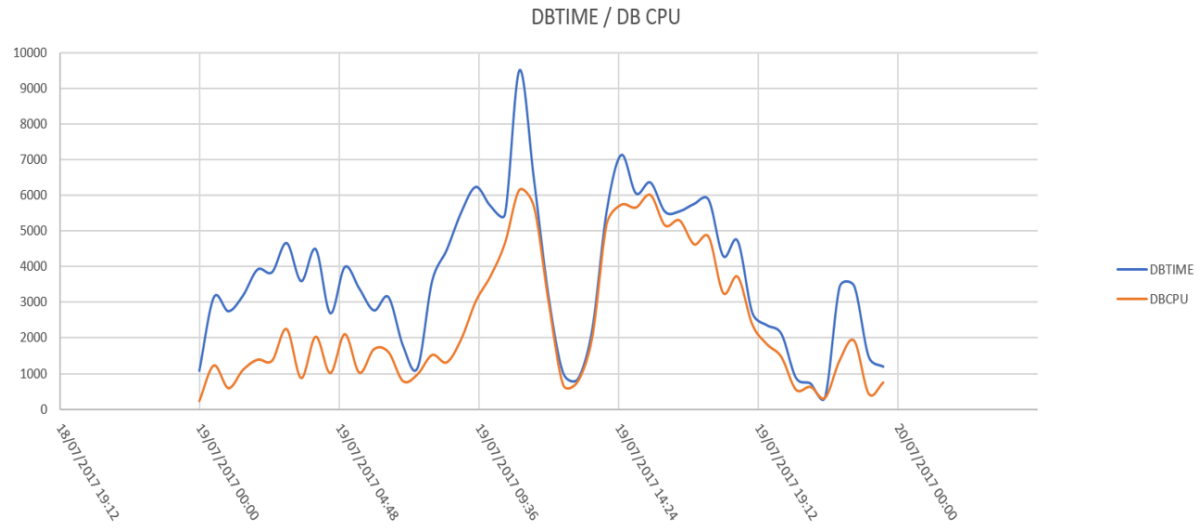

Illustration. 2: DB Time and DB CPU time visualization from statspack data.

What time reduction to target?

Once overload period, with AAS, Database Load, and DB Time, are identified, you can diagnose if, during these peak periods, the time is more represented by DB CPU Time or Active Wait Time.

The second step is to identify how the time is consumed during the peak period. Because, if the time is mainly represented by Active Wait Time, you will try to know which event is mainly represented during this peak period, and then you will try to reduce it to solve the issue (completely or partially).
If the time is mainly represented by DB CPU Time, maybe you will try to find SQL statement that touch tons of buffer or statements that run nested loops upon two huge data sets.

This next part of analysis can be done by visualizing time consumption during the peak period, or with the help of AWR or Statspack reports taken on this peak period.
All of these steps are the core of the drilldown approach and can be summarized in the figure below.

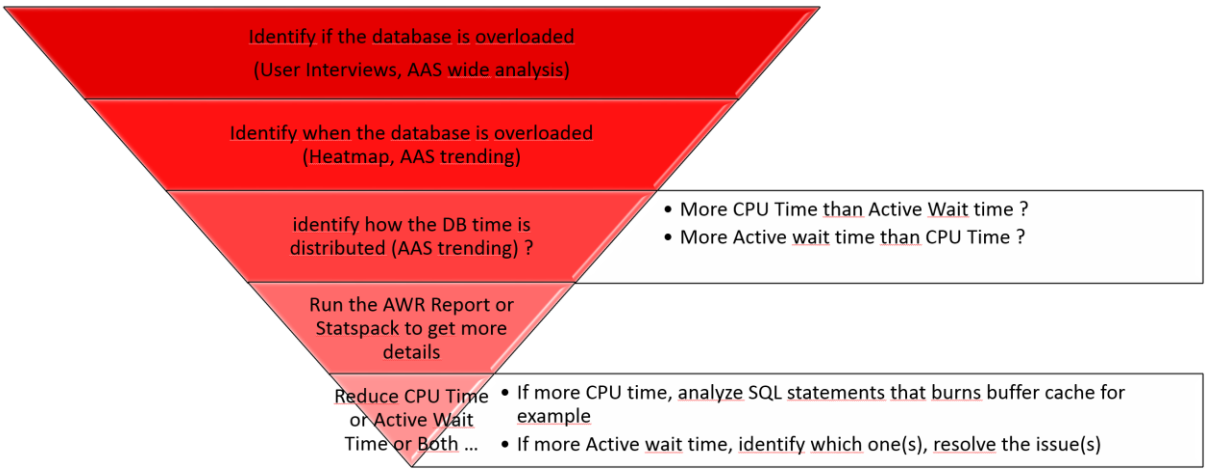

Illustration. 3: The drilldown approach summarized

Code instrumentation for a better analysis

For a better analysis and a quick identification of performance issues, it is highly recommended to instrument the application code executed on the database side.
This can be, Java code, PLSQL Code or every code that can call Oracle supplied PL/SQL packages.

When the code is correctly instrumented, information is added to dictionary views like V$SESSION, V$SQL_MONITOR etc.
Then, all these views are periodically snapped by AWR, ASH or Statspack and the data they have collected is archived on these repositories, including the new information collected by code instrumentation.

Information collected by code instrumentation is represented by application module (MODULE), action in the module (ACTION).
These module and action have to be set by the developer depending on which use case the application is executing.

For example, MODULE can be “PAYBACK EDITION” and the ACTION can be “PACKBACK HEADER”, “PAYBACK DETAIL”, “PAYBACK FOOTER”.

Code instrumentation is done through a supplied PL/SQL Package named DBMS_APPLICATION_INFO.

Below is an example of setting a module and an action inside a Java code sample (very basic and not optimized at all !).

```
String module;
String action;
Connection c = DriverManager.getConnection
("jdbc:oracle:thin:@192.168.99.8:1521:orcl", "system", "oracle");
CallableStatement call = c.prepareCall("begin
dbms_application_info.set_module( module_name => ?, action_name =>
?); end;");

module="PAYCHECK"; action="HEADER";

try{
    call.setString(1,module);
    call.setString(2,action);
    call.execute();
}
catch (SQLException e) {e.printStackTrace();}

// DO STUFF

module="PAYCHECK"; action="DETAIL";

try{
    call.setString(1,module);
    call.setString(2,action);
    call.execute();
}
catch (SQLException e) {e.printStackTrace();}
finally  {call.close();}

c.close();
```

Once instrumented and data collected, the load can be represented in several ways.
The best known is to use enterprise manager and to generate pie graphs. This is represented by the following figure. On this figure, we can see the difference between a well instrumented code, and a code that is not instrumented.

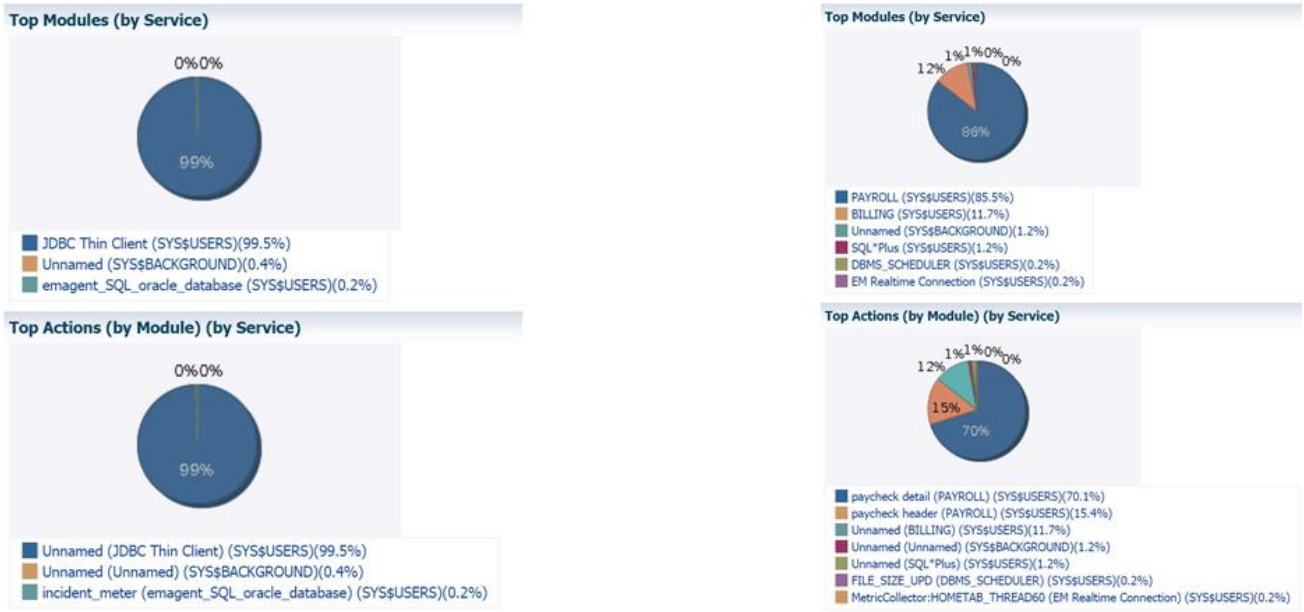

Illustration 3.: EM representation of a code that is not instrumented (Left) and a well instrumented code (right).

This representation can be done by yourself by querying ASH, AWR views and statspack tables and then visualize it in the tool of your choice (Excel, Tableau Software etc.).

PL/SQL profiling for a better analysis

For a better analysis, and especially when the problem seems to come from PLSQL programs, identifying bottleneck can be difficult.
A PL/SQL call (procedure, function or package) usually calls tons of other calls. It can include loops that become time consuming, even if the action in the loop is very fast. More, PL/SQL source includes SQL call and it can be interesting to detect context switching in it.

To identify all these types of bottlenecks, it can be useful to profile the PL/SQL code.

Profiling PL/SQL code needs very few installation steps: creating tables and Oracle directory structure. Identify the part of your code you want to profile, and add two instructions: start_profiling, stop_profiling.

PL/SQL profiler was introduced in Oracle 8i, with Oracle 11gR1 a new type of profiler has been added, a hierarchical profiler.
As said before, PL/SQL is something hierarchical, a procedure is called, this procedure calls a function etc.
This hierarchy is very useful to identify time consuming procedure at the first level, and then analyse the second level of hierarchy to identify the most time consumer in it, etc.
At the end of the analysis, the analyst or the dba can find the culprit and act to reduce the time.

In this part of the presentation, real use-case will be explained and we'll see how we can use flame graph tools to visualize a result of a profiling.

Contact address:

Laurent Leturgez
PREMISEO
7 rue Jules Guesde
59320 EMMERIN, FRANCE

Phone: +33 (0)6 60 99 70 46
Fax:
Email laurent.leturgez@premiseo.com
Internet: www.premiseo.com