Core Strategies for Trouble-Shooting

Jonathan Lewis
JL Computer Consultancy
UK

Keywords:

Optimisation Trouble-shooting Performance

Introduction

In an ideal world, everyone who had to handle performance problems would have access to ASH and the AWR through a graphic interface – but even with these tools you still have to pick the right approach, recognise the correct targets, and acquire information at the boundary that tells you why you have a performance problem and the ways in which you should be addressing it.

There are only three ways you can waste resources on an Oracle system, and only three different types of activity that need to be investigated. If you don't appreciate that this is the case then you can waste a lot of time following the wrong strategy and attempting to solve the wrong problems. Once you have a clear statement of what you are trying to achieve it becomes much easier to achieve those aims.

Three ways to cause problems

There really are only three symptoms you need to look out for in the database

- You're doing it the hard way
- You're doing it too often
- You're queueing / locking

Inevitably you can see that there is going to be some overlap between the three (and it would be easy to argue that the third is simply a side effect of the other two). If you are executing a single statement "the hard way" you're likely to be doing single reads or buffer gets too often. If you're executing a very lightweight statement too often it's probably a step in a process that is attempting get a job done "the hard way", and it's probably causing (and getting stuck in) queues for latches and mutexes. I've included queueing as the third option because the simple action of locking data (deliberately, or by accident) doesn't fall neatly into the other two.

Another way of looking at this list is to reduce it two just two items with a slightly different flavour: when you spend too much time on a task it's because you're either doing too much work, or you're not being allowed to work.

Three classes of problems to solve

An important aspect of performance problems is the high-level classification; labelling the class of problem properly points you to the correct strategy for investigating the problem. Again there are only three possibilities in the list, which I'll express as typical complaints:

- My report is taking too long to run / screen is taking to long to refresh
- The batch job over-ran the SLA last night
- The "system" is slow

What's the most significant difference between three classes of complaint ?

- "My XXX takes too long": XXX is a repeatable event that can be isolated and watched – just do it again, and again, and again, and again while I watch every step of what's going on.

- The batch job over-ran last night: It's not a repeatable process, so you've got to infer what the problem was from historical evidence; it's (usually) made up of a number of concurrent processes, which may interfere with each other to varying degrees depending on when their relative start and finish times are.

- The "system" is slow: possibly all the time, possibly intermittently – if there's no specific complaint then the only option is to key an eye open for resource-intensive activity to see if you can reduce the workload of specific individual tasks (for which read SQL or PL/SQL statements) or reduce the number of time that those tasks are executed.

The common source

Despite the fact that we can classify performance problems in three ways, it's worth remembering that ALL the information we might use to drive our diagnosis comes from one place – the Oracle database. At any moment we have sessions that are active, operating statements (cursors), and using a resource such as a file, a latch, a buffer, and so on. We could almost represent each moment by a cube, with sessions along one size, cursors along another, and resources along the third – the cube changes moment by moment, allowing us to visualise time as the fourth dimension in a hypercube of activity.

Instant by instant the Oracle kernel code knows which session is using what resource to operate which cursor – and although the total volume of all that information is far more than could reasonably be recorded, Oracle has many different ways of slicing and dicing that hypercube – running totals, rolling aggregates, snapshots by session, by cursor, by resource and so on – that are made visible as the dynamic performance views (v$ objects). Trouble-shooting is largely a case of deciding dynamic performance views are the most appropriate to use for our three classes of task.

Active Session History

Before reviewing the three classes, it's worth break off for a moment to say a few things about one of the most important and useful views that we have into the wealth of information available; this is the active session history (***v$active_session_history / ASH***) which Oracle uses to capture a snapshot once every second of each currently active session is doing; every 10th snapshot is then echoed down into the Automatic Workload Repository (AWR) by a process that runs every hour to copy the dynamic view to a table that can be view through the database view ***dba_hist_active_sess_history***.

The capture rates can all be adjusted: I have never seen anyone change from one snapshot per second, or every 10th snapshot in a production system, but I've often seen the dump to the AWR taken every 30 minutes, occasionally 20 or even 15 minutes. On occasion I have asked clients to do a CTAS (create table as select) to capture the whole of the v$active_session_history to a table that can be exported and reviewed at a later date. Oracle tries to keep a minimum of at least an hour's ASH in memory, and with a large enough SGA you may find that this will stretch out to 3 or 4 hours.

Unfortunately, although (or perhaps because) ASH and it's AWR history are extremely helpful, you have to pay extra licence fees to use the information, and the technology can only be licensed with the Enterprise Edition of Oracle.

My report is slow

The special feature of someone complaining about a specific task is that it's likely to be repeatable – so we can run it again and again and watch every single detail to see where the time goes. Our slice through the hypercube could take a single session over a period of time and report every action along that path. This, of course, is the 10046 – a.k.a extended SQL trace event. We can enable it in many ways, perhaps through a logon trigger, perhaps through a call to dbms_monitor:

```
begin
        dbms_monitor.session_trace_enable(
              session_id => &m_sid,
              serial_num => &m_serial,
              waits      => true,
              bind       => true,
              plan_stat  => 'all_executions'
        );
end;
/
```

In this example I've request all wait states and bind variable to be dumped into the trace file, I've also requested that the execution plan (with rowsource execution stats) be dumped for every single execution of every single statement. Sometimes a problem arises because a particular set of bind variables represents a special case that causes a "reasonable" plan to behave very badly. If we're going to look closely we may as well get as much detail as possible.

The entire "trace" interface was upgraded dramatically in 11g, and one of the useful variants on this them is particularly relevant to a common Web-based implementation. If you know that a specific screen task corresponds to a particular PL/SQL package you can enable tracing of a cursor (across the system, if necessary) by SQL_ID. So, for example, you might issue the following two commands, with a couple of minutes gap between the two:

```
alter system
set events 'sql_trace[SQL:1wthpj7as7urp]
              plan_stat=all_executions,
              wait=true, bind=true'
;

-- wait a few minutes

alter system
set events 'sql_trace[SQL:1wthpj7as7urp] off'
;
```

Every time the statement with SQL_ID ='1wthpj7as7urp' is executed, the session executing it will start adding information to the session trace file, and when the state ends the tracing will end. This is

particularly nice if the "statement" is a top-level call to a PL/SQL procedure because all the SQL inside the procedure will be traced as the package executes.

For a highly focused, highly repeatable task, the 10046 trace event is almost always all you need to do.

The batch over-ran

The big difference between this case and the previous one is that "the batch" is not something you can simply repeat and watch. Moreover, "the batch" is likely to be a large number of separate sections of code that are scheduled to run with a fairly fluid timetable that can result in changes from day to day (or, more likely, night to night) in the set of jobs that might be running concurrently. This means that even if you could re-run the batch job (perhaps on the previous night's backup – you might not see the same problem appear because a small change in timing could result in a large change in contention).

One of the most important steps of dealing with the batch is pre-emptive: instrument your code and make it possible to compare the run on one night with the run on another. At the very least you need to have something capturing the start and end times of each "significant component" of the batch so you can quickly answer questions like: "which jobs took much longer than usual", "which job was the first job that took longer than usual", "which jobs were running concurrently with job X last night when they never usually overlap?"

Ideally you should have much more information than this about each job – basically a report from Oracle which say "how much work did I do, how much time did I spend": for the session this is simply a report of v$sesstat (joined to v$statname) and v$session_event for the session; if you classify each job as "connect to the database, do something, disconnect" then this critical log is simple a pair of select statement spooled out somewhere convenient, or written to the database; if you want to break a single connection into significant pieces then a simple pl/sql procedure could read the statistics into a pl/sql array as the piece starts, then re-read the stats and calculate the differences as the piece ends.

Knowing where the time went, and know how tasks have behaved differently from previous runs is a big step forward to identifying the problem.

If you don't have the instrumentation you need then the AWR (if you're licensed) or Statspack (if you're not licensed) is a step in the right direction. Apart from the typical hourly snapshots and reports you can take a snapshot as the first and last steps of the batch so that you've got "the whole batch" in a single AWR report. If you've got that you can then do comparisons for things like:

- Which event consumed much more time than usual
- Which SQL took much more time than usual
- Which segment(s) saw much more activity than usual
- Was there some unusual enqueue activity
- Can we see some unusual outliers in the event histograms
- Can we see some unusual memory demands in the pga histogram

Although system-wide summaries rarely point us at exact causes, they can often give us strong clues of areas (and times) where problem originated.

In this respect the "Top Activity" screen from Enterprise Manager (Grid Control / Cloud Control) can be very helpful as it produces a nice graphical presentation of "business" – where, in the picture of last night's activity does the graph start to ramp up, and what colour is the one that's growing, and how

does that picture compare to the same picture the previous night. (Having two window open with two different nights make it wonderfully easy to switch between displays and spot the differences). Since the top activity screen is created from the dba_hist_active_sess_history, which contains about 100 different details per session of each captured moment, it's very easy to drill though the spikes to answer questions like: which object, what event, which SQL, what was the execution plan, how much work did that take, to follow the chain of time back to the cause.

The system is slow

If no-one is going to tell you about specific tasks, and if you don't have any sort of boundary that allows you to focus on tasks or time-ranges, then the simplest thing to do is look for anything expensive (i.e. time-consuming) and see if you can make it cheaper.

Again, the graphic "Top Activity" screen is very helpful, and I often tell people to arrange to have a system that shows the top activity screens for the most important 2 or 3 databases on a large screen on the wall where any passing DBA might notice a brief spike in workload. There are systems that can be improved by constant monitoring – so long as the monitoring doesn't take out 100% of an individual's time but is driven as an informal glance at a picture.

If you're not licensed to take advantage of the AWR then Statspack can help – but with the smallest time interval (though 15 minutes is as low as I've ever gone) between snapshots so that "anomlies" that are short-lived don't fall out of memory before they can be captured.

An important feature of reading Statspack is that you need to check for missing information – if the headline figure for physical reads is 25M but the "SQL ordered by reads" is 12M then you know that there must be 13M reads that didn't get captured in the report, and that might be the 13M that is causing the problem. Similarly if the "Segments by physical reads" reports 16M reads that's 4M more than the SQL – but is the 12M a subset of the 16M, or is there only a 3M overlap between the two figures so that between them the 12M and 16M cover the entire 25M. There's more information in the Statspack report than immediately meets the eye.

The other thing you can do with "the slow system" when you don't have ASH to help is take snapshots (or get some freeware in to do the same sort of thing). If the system is slow "right now" you could take a snapshot of v$sess_io (session I/O) wait 30 seconds and take another snapshot, find the diffrerence and see who is doing most of the I/O work – then chase that session; or take snapshots of v$sesstat limited to (say) statistics about "%redo%" and find out who is generating lots of redo.

Oracle allows you to take this approach back into history (to some degree) – there are a number of "metric" views which give you thing like the rolling average, min, and max I/O volumes for the last few intervals of 5 minutes or 1 minute each – telling you, in other words, whether there were any interesting bursts of extreme activity in the recent past. For example, a query against v$sysmetric_summary might give us an output like the following:

```
METRIC_NAME                     MAXVAL     AVERAGE STANDARD_DEV METRIC_UNIT
----------------------- ------------- ----------- ------------ ----------------------
Physical Reads Per Sec        1,618.95      105.92       358.16 Reads Per Second
Physical Reads Per Txn       97,202.00    5,539.19    20,811.56 Reads Per Txn
Redo Generated Per Sec    6,773,108.94  218,132.86 1,023,458.57 Bytes Per Second
User Calls Per Txn              395.00       43.39        79.85 Calls Per Txn
Total Parse Count Per Sec        31.14        1.88         4.25 Parses Per Second
Host CPU Utilization (%)         64.51        3.93         9.07 % Busy/(Idle+Busy)
Database Time Per Sec            82.96        6.65        15.37 CentiSeconds Per Second
I/O Megabytes per Second         35.58        2.62         5.73 Megabtyes per Second
```

This summarises the last 12 intervals of 5 minutes. If we look at "Physical Reads per Txn" we can see that there were some extreme swings in activity over that period, so we could drill down into v$sysmetric_history for "Physical Reads per txn", looking at the 1 minute granularity and see:

METRIC_UNIT	BEGIN_TIME	VALUE
Physical Reads Per Txn	05-feb 12:45:55	421.00
	05-feb 12:44:55	477.00
	05-feb 12:43:55	351.00
	05-feb 12:42:55	406.84
	05-feb 12:41:55	1,550.00
	05-feb 12:40:55	93,984.00
	05-feb 12:39:55	97,202.00
	05-feb 12:38:55	93,323.00
	05-feb 12:37:55	391.00
	05-feb 12:36:55	504.00
	05-feb 12:35:55	504.00
	05-feb 12:34:55	252.00

Yes, a few minutes ago something exploded onto the system doing a huge amount of I/O for about 3 minutes. If we're lucky we might now drill into the v$sesstat, or v$sess_io, or v$sessio_event to see if we can find a session that is responsible for a large amount of I/O; and then check v$open_cursor to see if it still has some open cursors that might (if we check v$sql) show us what caused the I/O.

When there are no specific complaints, we just keep an eye open for spikes in activity and try track them down as quickly and cheaply as possible to see if they're worth addressing.

Summary

Oracle gives you a huge amount of information about the work that's going on and the time that's being used in the database. Unfortunately the most useful repository of that information is in a dynamic performance view that can only be viewed in the Enterprise Edition after purchasing additional licences. However, the information is summarised, in many different ways in literally hundreds of other dynamic performance views, and it's easy to pick out helpful information from those views in a variety of ways.

Key to making the best use of those views, though, is recognising that different classes of performance problems require different strategies – and there are only three different classes of problems to worry about.

Contact address:

Name
JL Computer Consultancy
1 Saxonbury Gardens
Surbiton, Surrey, UK

Phone: +44(0)7973 188785
Email info@jlcomp.demon.co.uk
Internet: http://jonathanlewis.wordpress.com