Core Strategies for Trouble-shooting

Jonathan Lewis

jonathanlewis.wordpress.com

www.jlcomp.demon.co.uk

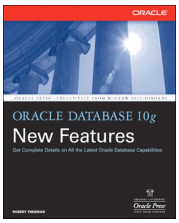
Who am I ?

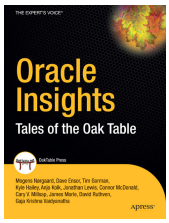
Independent Consultant

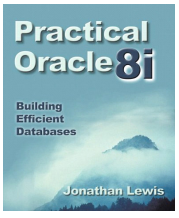
32+ years in IT
27+ using Oracle

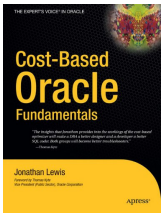
Strategy, Design, Review, Briefings, Educational, Trouble-shooting

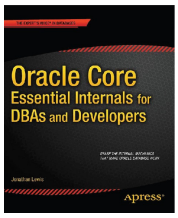
Oracle author of the year 2006
Select Editor's choice 2007
UKOUG Inspiring Presenter 2011
ODTUG 2012 Best Presenter (d/b)
UKOUG Inspiring Presenter 2012
UKOUG Lifetime Award (IPA) 2013
Member of the Oak Table Network
Oracle ACE Director
O1 visa for USA











The Target

- Minimise effort / maximise return

The Implementation

- Recognise the problem
- Pick the right strategy

Three ways of causing problems

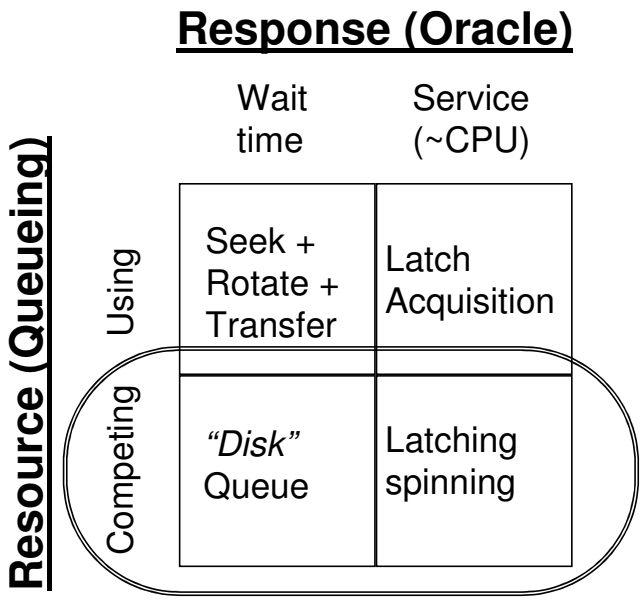
The code is doing it the hard way

The code is doing it too often

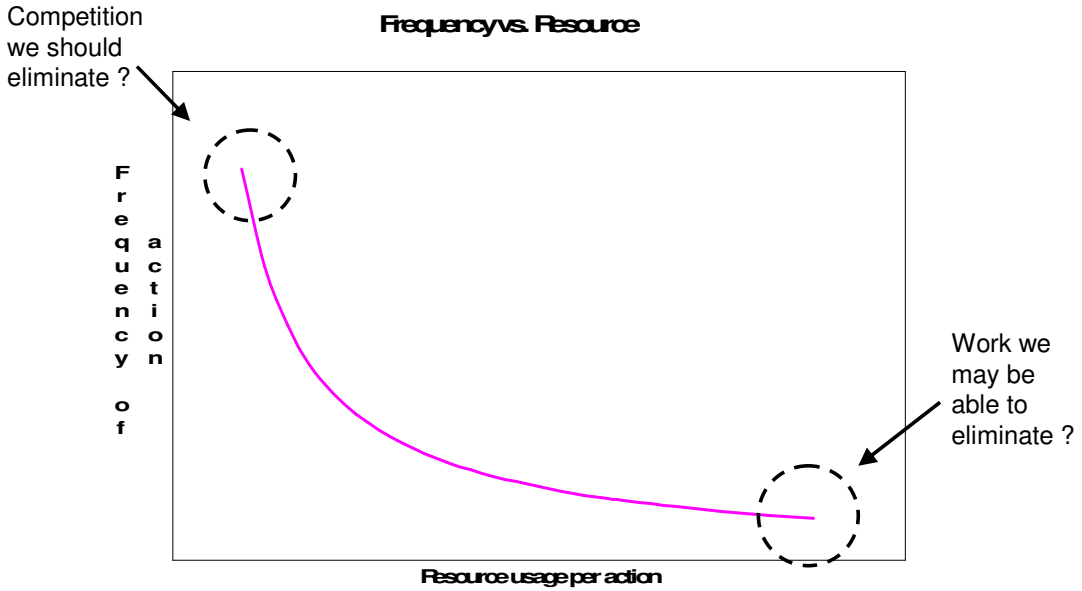
The code is causing locks / queues

Very often the 3rd is a side effect of the other two; and you could argue that if you're doing the first you're just doing the second at a finer granularity

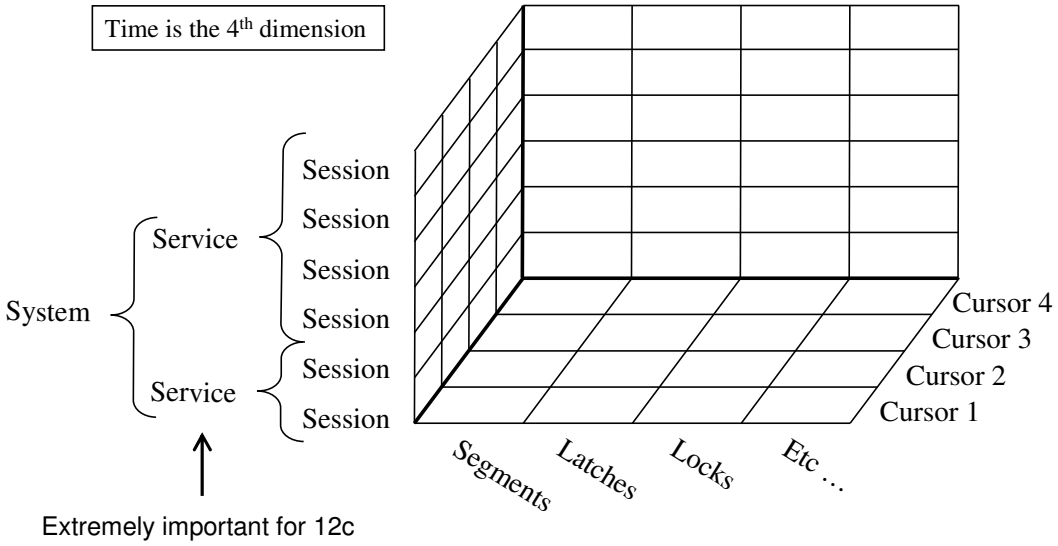
Response time = Service time + Wait time



The obvious two threats



Measures in four dimensions



Three classes of complaint

My task (screen, report) is slow

The batch took too long last night.

"The system" is slow

- Always
- Intermittently

Three strategies

My task (screen, report) is slow

- Easy to keep repeating task (usually)
- 10046, level 8 or 12 – one task, all details (*dbms_monitor*)
- Lots of options from 10g onwards for viewing specific activity

The batch took too long last night.

- Lots of sessions, not easily repeatable
- End of job summaries – overview only
- *ASH (if licensed) a historic **sample** of "10046"*
- *Monitor Database Operations (12c, if licensed)*

"The System" is slow

- AWR / Statspack – overview only, resources and SQL
- Sampling – one SQL / Session / Resource
- "metric" views - v$sysmetric etc.

Follow that session

```
alter session set events '10046 trace name context forever, level 12';

Level 4 => bind values        Level 8 => Wait events
Level 16 => dump plan on every execution
Level 32 => don't dump plans
Level 64 => dump plans intermittently as resource usage accumulates

 begin
        dbms_monitor.session_trace_enable(
                session_id => &m_sid,
                serial_num => &m_serial,
                waits      => true,
                bind       => true,
                plan_stat  => 'all_executions'
        );                      -- 'never', 'first_execution'
 end;                           -- no equivalent to level 64
 /
```

Precision Targets (a)

```
alter system
   set events 'sql_trace[SQL:1wthpj7as7urp]
       plan_stat=all_executions,
       wait=true, bind=true'
;
--       http://tech.e2sn.com/oracle/troubleshooting/oradebug-doc

-- wait some time  (about 2 minutes on a recent client case)

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

tkprof xxx_ora_12345.trc temp.prf sort=\(exeela, fchela\)
```


Precision Targets (b)

```
SELECT TO_CHAR(TO_DATE(:B3 ,:B2 ,'NLS_CALENDAR=''GREGORIAN'''),:B1 )
FROM DUAL


call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.00       0.00          0          0          0           0
Execute    930      0.56       0.53          0          0          0           0
Fetch      930      0.00       0.01          0          0          0         930
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     1862      0.56       0.55          0          0          0         930
```

I had the bind variables in the trace file:

Half the calls had a value for :B3 of '01-01-1000', the other half had '31-12-4172'

Precision Targets (c)

```
SELECT *
FROM    TABLE_X
WHERE   CODE_HOST = :B2
AND     TRUNC(SYSDATE)
            BETWEEN NVL(BEGIN_DATE,
                TO_DATE(
                        UTILS.NORMALISE_DATE('01-01-1000', 'DD-MM-YYYY'),
                        UTILS.GET_NLS_DATE_FORMAT
                )
            )
            AND NVL(END_DATE,
                TO_DATE(
                        UTILS.NORMALISE_DATE('31-12-4712', 'DD-MMYYYY'),
                        UTILS.GET_NLS_DATE_FORMAT
                )
            )                           -- why not just
                                        -- to_date('01-01-1000','dd-mm-yyyy')
                                        -- to_date('31-12-4712','dd-mm-yyyy')
```


The calls to ***dual*** were the effect of a pl/sql function in the ***where*** clause.
Fixes: fixed value, deterministic functions, or *(select function() from dual)*.

Slow batch – a

"The batch over-ran by 3 hours last night. Make sure it doesn't happen again tonight"

Best strategy: Pre-emptive reporting

- Statspack *(AWR)* snapshots before and after
- Compare two batches with *awrddrpt.sql*
- ASH (dba_hist_active_sess_history – *ashrpt.sql*)

Always helpful to know the code and business.

Every batch job could end with two queries to show the work done and the time lost in the process; then you can compare good behaviour against bad.

Slow batch – b

```
select  stn.name, mst.value                                  -- work done
from    v$mystat mst, v$statname stn
where   mst.value != 0
and     mst.statistic# = stn.statistic#

select  event, total_waits, total_timeouts,                  -- time waited
        round(time_waited/100,2)      time_waited,
        round(max_wait/100,2)         max_wait
from    v$session_event
where   sid = {current session id}

 select  stat_name, value                                    -- time location
 from    V$sess_time_model
 Where   sid   = {current session id}
 and     value != 0
```


A few bits of SQL at the end of each task can indicate why a program had a problem. 10g gave us ***v$sess_time_model*** as a summary how time is spent

Slow batch – c

Good

NAME	VALUE
session logical reads	151,302
CPU used by this session	2,744
DB time	3,054
physical reads	3

NAME	WAITS	WAIT_SEC	AVG_CS	MAX_CS
db file sequential read	3	0.02	.57	5

Bad

NAME	VALUE
session logical reads	151,302
CPU used by this session	3,038
DB time	19,924
physical reads	79,184

NAME	WAITS	WAIT_SEC	AVG_CS	MAX_CS
db file sequential read	79,184	166.29	.21	5

30 sec

3:19

This is a job that runs regularly – but the performance is erratic even though the work done is constant. It's critically dependent on caching benefits.

Slow batch – d

NAME	VALUE	
session logical reads	151,302	
CPU used by this session	3,401	
DB time	88,858	***14:50***
physical reads	***139,623***	

NAME	WAITS	WAIT_SEC	AVG_CS	MAX_CS
db file sequential read	139,623	851.70	***.61***	5

Variation in performance:

0:30	Everything is in the Oracle cache
3:20	A lot of data is read by Oracle, but in the SAN cache
14:50	Most blocks have to come from the SAN – with disk reads
(not shown)	It's all on disc, and the discs are overloaded.

This might be a (rare) example where taking advantage of the KEEP cache could stabilise key reporting times without disrupting other jobs.

The system is slow

No specific target

No consistent complaint

How do you find something to fix ?

Metrics (a)

```
select  {columns}
from    v$sysmetric_summary
order by
        metric_id
;
```

```
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
```

For a metric that shows an unexpectedly large standard deviation you can drill down to ***v$sysmetric_history*** to get the individual figures for each minute.

Normally you can expect ***v$sysmetric_summary*** to cover 61 intervals of 60 seconds each. There are 135 stats - you might choose a meaningful handful.

Metrics (b)

```
select  {columns} from v$sysmetric_history
where   metric_name = 'Physical Reads Per Txn'
and     group_id = 2            -- 60 second interval
order by begin_time desc;
```

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

The OEM interface alerts you to extremes variations. For usage notes see: http://oracledoug.com/serendipity/index.php?/plugin/tag/adaptive+thresholds

Sampling (a)

```
create or replace package body snap_sess_io as
-- some global cursors and variables here
cursor c1 is
        select  physical_reads, optimized_physical_reads, block_gets,
                block_changes, consistent_gets, consistent_changes, sid
        from    v$sess_io;

type w_type is table of c1%rowtype index by binary_integer;
w_list          w_type;

procedure start_snap is
begin
        for r in c1 loop
                w_list(r.sid).block_gets     := r.block_gets;
                w_list(r.sid).block_changes  := r.block_changes;
                ...
        end loop;
end start_snap;
```


If you don't have the licences to use the performance and diagnostic packs, you can write simple code to take very short snapshots. I use sys packages.

Sampling (b)

```
procedure end_snap is  -- lots of little bits cut out.
   for r in c1 loop
        if (
                (w_list(r.sid).block_gets != r.block_gets)
            or ...
        ) then
                dbms_output.put(rpad(r.sid,6));
                dbms_output.put(to_char(
                        r.block_gets - w_list(r.sid).block_gets,
                                '9,999,999,990'));
                ...
                dbms_output.new_line;
        end if;
   end loop;
end;
```

See also: http://blog.tanelpoder.com/files/scripts/snapper.sql

Within the end-snapshot routine, we have to handle missing rows from the array, and then do the arithmetic and output. ***utl_file*** would also be good.

Sampling in 12c (a)

```
with
        function wait_row (i_secs number, i_return number)
        return number is
        begin
            dbms_lock.sleep(i_secs);
            return i_return;
        end;
select
        event, wait_time_milli, sum(wait_count) wait_count
from    ({inline view})
where   wait_time_milli != -1
group by
        event, wait_time_milli
order by
        event, wait_time_milli
;
```


Sampling in 12c (b)

The inline view

```
select
        event, wait_time_milli, -1 * wait_count wait_count
from    v$event_histogram
where   event = '&m_event'
union all
select
        null, wait_row(10, -1), null
from    dual
union all
select
        event, wait_time_milli, wait_count
from    v$event_histogram
where   event = '&m_event'
```

Sampling in 12c (c)

```
define m_event = 'db file sequential read'

EVENT                                WAIT_TIME_MILLI   WAIT_COUNT
---------------------------------- --------------- ------------
db file sequential read                          1          411
db file sequential read                          2           11
db file sequential read                          4            4
db file sequential read                          8           33
db file sequential read                         16           37
db file sequential read                         32           10
db file sequential read                         64            5
db file sequential read                        128            0
db file sequential read                        256            0
db file sequential read                        512            0
db file sequential read                      1,024            0
```


Summary

- Look for high cost AND high frequency
- The problem can be competition
- All the measures come from one place
- Three problems == Three strategies