Yet Another Presentation on Extended SQL Trace

Richard Foote

Objectives

- Outline why “conventional” performance diagnosis techniques are flawed
- History of Wait Event Instrumentation
- Introduce the “Response Time” tuning methodology
- Describe how to enable extended SQL tracing
- Describe how to interpret a raw extended SQL trace
- 10g Wait Event new features

Where's The Milk ?

- Asked my wife to pop down to the local shop to buy some milk. A full 60 minutes later, she finally returns and pops the milk on the kitchen bench.
- This "response time" is clearly unacceptable, I had expected the milk in 5 minutes !!
- So how can I improve the response time ?
- What would be my first question … ?

DBA Cry For HELP !!

Extract of a post on comp.database.oracle.server newsgroup:

"I have a problem with an Oracle 8.1.5 database in that recently the database seems to be running very slowly. I have tried:

1) Rebuild indexes

2) Analyze NT performance

3) Checking fragmentation

I spent a couple of days on the net trying to find solutions but have not had much success…"

Unfortunately, unless we know what Oracle is doing and what it's spending its time on, any diagnosis would be pure guesswork …

“Conventional” Tuning Methodology

The “database” or “application” or “database users” are experiencing performance problems. It’s running slooooooooowly.

Step 1: Take out long checklist of potential problems

Step 2: Start with item 1, see if it’s the problem (typically related to some ratio or another using statspack as source)

Step 3: Hope problem is solved after tinkering with item

Step 4: If not, move onto next item on the list

Step 5: Repeat process until problem is eventually solved, just goes away or users stop complaining …

Problems with Conventional Tuning Methods

- Is very much a hit ‘n’ miss affair
- Is often based on inappropriate “scoping” of diagnosis data
- Can take significant amount of wasted time/resources to resolve
- Can lead to fixing “problems” that don’t actually directly impact performance
- Can lead to confusion as to what eventually fixed problem
- Can lead to confusion as to what the problem was
- Can lead to confusion as to whether the problem has actually been solved …

Users are a funny bunch …

- They don’t care that the database appears to be running well
- They don’t care your count(*) from dba_tables runs really really fast
- They don’t care other users aren’t complaining
- They don’t care all hit rates and ratios are really really high
- They don’t care it’s a tricky one and that you’ve gone through your checklist without luck
- They don’t care all is green on your Quest Spotlight screen
- When it comes to performance, they only care about …

RESPONSE TIMES !!

Response Time

Response times can simplistically be broken up into two basic components:

Time it takes **doing** something

+

Time it takes **waiting** on something

Anjo Kolk and Co in their groundbreaking "Yet Another Performance Profiling Method (YAPP)" paper denote response time as:

Response Time = Service Time + Wait Time

Problem with standard tracing is that wait time is unknown…

Birth of Extended SQL Trace

- In 1992 Oracle development lead by Juan Loaiza were having problems solving a problem using “conventional” methods
- Response times were bad but they had no idea where Oracle was spending its time
- In desperation, they decided to embed wait instrumentation into database code
- They could then clearly determine where time was being spent and problem was immediately found
- Extended SQL tracing was born…

Evolution Of Wait Event Interface

- V7:106 wait events
- V8:215 wait events
- V9:399 wait events
- V10:808 wait events

Note: Various enhancements in each release to be discussed…

List of ‘Common’ Events

db file sequential read
db file scattered read
db file parallel read
db file parallel write
direct path read
direct path write
direct path read (lob)
direct path write (lob)
log file switch completion
log file sync

control file sequential read
control file parallel write
cpu service
buffer busy waits
free buffer waits
enqueue
latch: library cache
latch: library cache pin
latch: redo allocation
latch: shared pool

…….

Where to Find Wait Event Info

- numerous v$ views, eg:
  - v$system_event (aggregates database stats)
  - v$session_event (aggregates session stats)
  - v$session_wait (current session waits)
  - v$session_wait_history (last 10 wait events/session)
- bstat/estat, Statspack, ADDM / AWR (10g) …
- But there's a major drawback with many of the above to diagnose specific tuning issues …

Problem with “Generalised” Wait Event Info

- Specific tuning problem can be “drowned” out by all other activities
- Once data is aggregated, specific information is lost
- Averages can be very misleading (database/session)
- Database level statistics don’t necessarily translate to a specific performance issue
- Fundamentally, aggregated statistics don’t disclose where time is spent during specific tuning problems

Scoping is Vital

- Firstly, we ideally want to focus and prioritize on sessions/applications important to business
- Ideally we need to capture where time is spent just for the session experiencing performance problems
- Ideally we need to capture where time is spent just for the duration of the performance problem
- Any more and aggregation can be misleading and introduce discrepancies
- Any less and vital data could be missing
- So how can we get just the right amount of data …

EXTENDED SQL TRACING !!

Some Pre-Requisites First …

- timed_statistics = true
- max_dump_file_size=unlimited

  obvious risk but else risk miss vital info …
- user_dump_dest=location of trace file (user)
- background_dump_dest=location of trace file (background)
- set tracefile_identifier ='trace_tag'

How to enable Extended SQL Trace

alter session set events ‘10046 trace name context forever, level 12’;

where level can have the following levels:

1 - standard sql trace (as per sql_trace and the such)

4 - include bind values in trace file

8 - include wait events in trace file

12 - include both bind and wait events in trace file (recommended)

alter session set events ‘10046 trace name context off’;

Trace Another Session

To trace another session (from version 7):

dbms_system.set_env(sid, serial#, 10046, 12, ‘’)

dbms_system.set_env(sid, serial#, 10046, 0, ‘’)

Note: a little dangerous due to risk of entering incorrect event …

Enabling Extended SQL Trace

Version 8: dbmssupp.sql

dbms_support.start_trace(true, true)

dbms_support.stop_trace()

dbms_support.start_trace_in_session(sid, serial#, binds=> true, waits=>true)

dbms_support.stop_trace_in_session(sid, serial#)

Version 9i:

dbms_session.session_trace_enable(true,true,'ALL_EXECUTIONS')
dbms_session.session_trace_disable()

Version 10g:

dbms_monitor.session_trace_enable(sid, serial#, waits=>true, binds=>true)

dbms_monitor.session_trace_disable(sid, serial#)

Raw Trace File: Preamble

```
Dump file c:\oracle\admin\floyd\udump\floyd_ora_2788.trc
Fri Aug 06 19:07:32 2004
ORACLE V10.1.0.2.0 - Production vsnsta=0
vsnsql=13 vsnxtr=3
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options
Windows XP Version V5.1 Service Pack 1
CPU            : 1 - type 586
Process Affinity: 0x00000000
Memory (A/P)    : PH:92M/510M, PG:743M/1249M, VA:1849M/2047M
Instance name: floyd
Redo thread mounted by this instance: 1
Oracle process number: 14
Windows thread id: 2788, image: ORACLE.EXE (SHAD)
```

Raw Trace File: Session Details

```
*** ACTION NAME:() 2004-08-06 15:07:32.613
*** MODULE NAME:(SQL*Plus) 2004-08-06 15:07:32.613
*** SERVICE NAME:(SYS$USERS) 2004-08-06 15:07:32.613
*** SESSION ID:(150.35) 2004-08-06 15:07:32.613
```

Raw Trace File: Parsing in Cursor

```
PARSING IN CURSOR #9 len=98 dep=0 uid=28 oct=3 lid=28 tim=8526767622
hv=404259818 ad='7afad994'

select count(*) from bowie , ziggy where bowie.table_name=ziggy.segment_name and
bowie.owner='SYS'

END OF STMT
```

len: length of sql text

dep: recursive depth of sql

uid: user id (who parsed sql)

oct: Oracle command type

lid: Privilege user id (eg. owner of calling procedure)

tim: Timestamp (timings in microsec 9i and above, previously centisec)

hv: Hash id

ad: SQLTEXT address (as in v$sql)

Raw Trace File: Database Calls

```
PARSE #9:c=46875,e=490772,p=3,cr=102,cu=0,mis=1,r=0,dep=0,og=1,tim=8526767615
```

PARSE, EXEC, FETCH, UNMAP, SORT UNMAP, ERROR operations

#:	cursor number
c:	cpu time
e:	elapsed time
p:	physical read requests
cr:	consistent reads
cu:	current reads
mis:	cursor misses in library cache
r:	rows processed
dep:	depth of cursor (0 = user level, >0 = recursive)
og:	optimizer goal (1=all_rows, 2=first_rows, 3=rule, 4=choose)
tim:	timestamp

Raw Trace File: Binds

```
BINDS #9:
bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=08 oacfl2=0001 size=24 offset=0
   bfp=05536990 bln=22 avl=02 flg=05
   value=42
```

bind n: bind position (note equates to :bn+1 in sql)

dty: data type

mxl: maximum length

mal: array length

scl: scale

pre: precision

oacflg: various bind option flags

size: memory size

offset: offset into bind buffer

bfp: bind address

bln: bind buffer length

avl: actual length

flag: bind status flag

value: actual bind value

Raw Trace File: Wait Events

```
WAIT #9: nam='SQL*Net message to client' ela= 6 p1=1111838976 p2=1 p3=0
WAIT #9: nam='db file scattered read' ela= 18816 p1=4 p2=1810 p3=8
WAIT #9: nam='db file sequential read' ela= 353 p1=4 p2=1821 p3=1
```

nam: name of wait event

ela: elapsed time of wait

p1: parameter 1 for specific wait event (eg. file# - db file scattered read)

p2: parameter 2 for specific wait event (eg. block# - db file scattered read)

p3: parameter 3 for specific wait event (eg. blocks – db file scattered read)

It's this level of detail that describes the difference between CPU and overall response times !!

Raw Trace File: Better Format

```
WAIT #1: nam='db file sequential read' ela= 10584 file#=8 block#=1033 blocks=1
obj#=78294 tim=615409524250
WAIT #1: nam='db file scattered read' ela= 13980 file#=8 block#=1034 blocks=16
obj#=78294 tim=615409551520
WAIT #1: nam='db file scattered read' ela= 1487 file#=8 block#=1050 blocks=16
obj#=78294 tim=615409554677
```

nam: name of wait event ela: elapsed time of wait

file#: file number of data file accessed

block#: block id of first block accessed in I/O operation

blocks: number of blocks read by I/O operation

obj#: Database object id of database object being accessed

In later versions of Oracle, wait info is more detailed and easier to read

Raw Trace File: Row Source Operations

STAT #9 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=938 pr=108 pw=0 time=331605 us)'

STAT #9 id=2 cnt=328704 pid=1 pos=1 obj=0 op='HASH JOIN (cr=938 pr=108 pw=0 time=1790903 us)‘

STAT #9 id=3 cnt=9728 pid=2 pos=1 obj=15054 op='TABLE ACCESS FULL BOWIE (cr=562 pr=108 pw=0 time=34953 us)'

STAT #9 id=4 cnt=67472 pid=2 pos=2 obj=15057 op='INDEX FAST FULL SCAN ZIGGY_IDX (cr=376 pr=0 pw=0 time=202491 us)‘

id: row source id

cnt: no of rows returned

pid: parent id

pos: position within same parent level

obj: object id

op:operation

cr: consistent reads

pr: physical reads

pw: physical writes

time: elapsed times

TKPROF and Extended Trace

- Since 9i, TKPROF can also format and summarise wait details in trace file
- Bind data however is not displayed
- In many cases, can simplify and ease interpretation of trace data
- But, can hide important details
- Can have issues with double counting

TKPROF: Standard SQL Trace

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.05	0	0	0	0
Execute	1	0.09	**982.84**	615	1714	22	48
Fetch	0	0.00	0.00	0	0	0	0
total	2	0.10	**982.89**	615	1714	22	48

Rows Row Source Operation

---

0 UPDATE (cr=1714 pr=615 pw=0 time=**982840119** us)

48 TABLE ACCESS FULL BOWIE (cr=1713 pr=613 pw=0 time=2715035 us)

Classic example : why the massive difference between CPU time and elapsed time ?

Let's check out Database Level Statspack

Top 5 Timed Events

~~~~~~~~~~~~~~~~~~~~

Event	Waits	Time (s)	% Total Ela Time
db file sequential read	943,457	18,678	46.33
db file scattered read	381,532	6,059	15.03
CPU time		5,627	13.96
direct path read (lob)	326,048	5,550	13.77
SQL*Net more data to client	204,957	3,051	7.57

It certainly looks like we might have an I/O related problem here …
~~~~~~~~~~~~~~~~~~~~

TKPROF: Extended SQL Trace

Elapsed times include waiting on following events:

Event waited on	Times Waited	Max. Wait	Total Waited
db file sequential read	565	0.31	2.70
db file scattered read	18	0.01	0.07
enq: TX - row lock contention	**319**	**3.12**	**979.65**
SQL*Net message to client	1	0.00	0.00
SQL*Net message from client	1	9.16	9.16

****

With extended SQL wait data, we're lead to the actual cause of the problem

Note also the max Times Waited is not always the actual problem …

Oracle Response Times

Millsap/Holt break response times with two different categories:

- Time within database calls (i.e. database in the process of performing a task)
- Time between database calls (i.e. database waiting to be asked to perform a task)

Time Within Database Calls

Elapsed time during a database call is approx:

e = c (cpu time) + sum (ela within call)

Approx. due to unaccounted times and double-counting

Note: waits times for a database call are all listed directly before the database call

Time Between Database Calls

- Can be simply expressed as:

  Sum (ela) times between database calls

- Examples of between (idle) database calls:
  - ✓ SQL*Net message to client
  - ✓ SQL*Net message from client
  - ✓ pmon timer
  - ✓ smon timer …
- Do not ignore (as often stated) so-called “idle” database calls …

Putting Total Response Times Together

Total response time for a traced file is approx:

r=Time Within Calls + Time Between Calls

r=c (cpu) + sum (elapsed times within calls) + sum (elapsed times between calls)

r= c (cpu) + sum (ela)

Careful of double counting

- Recursive SQL is also listed in the trace file
- Identified with a dep > 0
- Child recursive calls are listed immediately before the parent (child dep – 1)
- e = e (dep 0) – e (dep > 0)

- CPU can also be double counted as some ela timings are inclusive of CPU (generally insignificant)

Tracing Parallel Execution

- Statements running in parallel spawn Px processes
- These processes generate their own individual trace files
- All related trace files need to be studied and considered
- Not too difficult to achieve

Tracing Multi-tier Environments

- Very difficult to consolidate trace data
- Client sessions can share and use multiple server sessions
- Session trace data can be spread across multiple trace files
- Multiple sessions can be intermixed into a single trace file
- Most solutions required altering temporarily how applications are invoked
- 10g has simplified things …

10g New Extended Trace Features

- Perhaps most important is that all is now fully supported and documented by Oracle
- dbms_monitor package
- No. of events increased from 399 to 800+
- Lots of new wait related v$views, including:
  - v$session_wait_class
  - v$session_wait_history
  - v$sess_time_model
  - v$service_event
  - v$service_wait_class

10g Wait Events New Features

- Can now trace data based on new criteria:
  - Client Identifier
  - Service Name
  - Module Name
  - Action Name
- Developers Note !! End client identifying info can be set via:
  - dbms_session
  - dbms_application_info
  - JDBC/OCI call attributes
- Allows developers to label instrumentation parameters in applications
- Examples:
  - exec dbms_session.set_identifier('Bowie');
  - OCIAttrSet (session, OCI_HTYPE_SESSION, (dviod *) "bowie", (ub4)strlen("bowie"), OCI_ATTR_CLIENT_IDENTIFIER, error_handle);

Statistic Aggregation Dimensions

- Statistics can now be aggregated across these new dimensions
- dbms_monitor.client_id_stat_enable('bowie)
- dbms_monitor.serv_mod_act_stat_enable('service_name','module_name','action_name',waits, binds')
- New views include:
  - dba_enabled_aggregations
  - v$client_stats
  - v$service_stats
  - v$serv_mod_act_stats
- However, remember problems with aggregating data …

End-To-End Tracing With 10g

- Extended tracing can also be globally enabled across new dimensions
- Simplifies tracing of specific issues in n-tier and shared server environments
- dbms_monitor.client_id_trace_enable('bowie', waits=>true, binds=>true)
- dbms_monitor.serv_mod_act_trace_enable('service_name','module_name','action_name',waits, binds)
- New view: dba_enabled_traces

Note: OEM fully supports all new features discussed

TRCSESS Command

- Simplifies consolidation of trace data in n-tier and shared server environments
- Generates a single trace file by extracting required data from specified trace files
- **trcsess output=output.trc client_id='bowie' trace_file_1.trc trace_file_2.trc**

Warnings Put In Perspective

- Pre-10g, extended SQL trace was not “officially” promoted by Oracle (despite numerous metalink articles)
- Careful what and how much you trace, can consume much disk
- Writing to disk can impact overall response time
- Version 9 collects stats at the rowsource level in plans, can impact CPU consumption
- Like any feature, can have bugs (eg. 3009359 which causes excessive cpu overheads in 9.2.0.4)
- However, consider “cost” of not correctly diagnosing performance issue …

Further Reading

- www.hotsos.com: numerous excellent articles
- www.oraperf.com: YAPP and other articles
- Metalink has numerous notes:
  - 21154.1 10046 enable SQL statement tracing
  - 39817.1 Interpreting raw SQL trace output
  - 62294.1 The DBMS_SUPPORT package
  - 242374.1Tracing PX session with a 10046 event

A Must Read For Every Oracle DBA

- Cary Millsap and Jeff Holt – “Optimizing Oracle Performance”

- Excellent !!