ADVANCED PERFORMANCE TUNING USING EVENT WAITS INTERFACE

E. BÂRLĂDEANU*

Marile întreprinderi actuale folosesc, pentru managementul resurselor proprii, sisteme informatice complexe integrate, de tip ERP (Enterprise Resource Planning). În cursul exploatării lor de către un număr mare de utilizatori, apar situații când unele componente ale performanței (şi în special timpul de răspuns) sunt serios afectate şi necesită o acțiune rapidă şi eficientă din partea administratorilor. Metoda prezentată mai jos, preluată ca idee din mediul economic constă în detectarea şi rezolvarea problemelor în ordinea gravității lor, urmând ca problemele mai puțin importante să fie minimizate (sau chiar rezolvate) de la sine (efectul "trickle down"). Articolul se referă la configurația SAP R/3, ORACLE, Solaris, VxVM, Symmetrix, dar metoda poate fi extinsă şi la alte ERP.

Large enterprises nowadays use complex integrated information systems for the management of their own resources: ERP (Enterprise Resource Planning) systems. During their use by a considerable large number of users, there are various situations when some perfomance components (such as response time) are seriously affected and require quick and efficient actions from the administrators.

The method presented below, based on principles from the economic field, consists of detecting and solving the problems in order of importance, the less important problems being minimized (or solved) by themselves (the "trickle down" effect). It can be used for other ERP systems too. The paper is based on a SAP R/3, ORACLE, Solaris, VxVM, Symmetrix configuration, but the method can be extended.

Keywords: event waits interface, response time, database bottleneck, "trickle down" effect.

Index of Acronyms: CBO - Cost **B**ased **O**ptimizer**; DBA** – **D**ata**B**ase **A**dministrator; **DBWR** - **D**ata**B**ase **Wr**iter; **ROI** - **R**eturn-**O**n-**I**nvestment; **VxVM** – Veritas Storage Software Management [7]; **Symmetrix** - an EMC^2 networked storage solution [10].

Introduction

To oversimplify, the *event waits interface* is a set of dynamic performance views and trace files where Oracle [4] constantly instruments its performance

* Eng, SAP R/3 Senior Technical Consultant; Bucharest, ROMANIA.

metrics broken down by session as well as aggregated metrics across the database instance as a whole.

These metrics are expressed in terms of how much time a specific session/the entire system has spent [1, 2] walking down each section of codepath inside the Oracle kernel (Codepaths have been given descriptive names by the Oracle kernel developers, and those names are also being referred to as "events").

Querying the wait interface enables the DBA to generate session-level and instance-level resource consumption profiles and find out useful tuning information such as what the number one time-consuming activity/codepath was for a given session or the database instance as a whole.

Knowing how much time each session is waiting on a given event will make it possible to predict by how many seconds the response time will get better if the bottleneck suggested by the name of the event is removed.

This is essential to know how to do if the enterprise is planning on allocating financial resources to fix the performance problem.

1. Tuning by event waits

We will take a close look at a sample resource profile and walk through the steps involved in the tuning process, all the way to the point where we have translated relevant performance metrics and event names into recommended performance relief actions.

A script [5, 4] based on the above presented configuration is used to generate the profile.

Essentially, the output of the script shows which performance metrics Oracle has updated within the last 10 seconds and by how much.

```
$ cat evtwaits
#!/bin/ksh
#Resource Profiler - shows which performance metrics Oracle has updated within the last 10 seconds and
#by  how much.
touch znew02.txt
while true
do
mv znew02.txt zold02.txt
sqlplus "/ as sysdba" << EOT > /dev/null
set pagesize 0 linesize 500 term off feedback off
spool znew01.txt
```

```
select
  to_char(sid) || '_' || replace(event,' ','_') event, time_waited
from
  v\$session_event
where
  time_waited > 0
order by
  event, sid;
spool off
EOT
clear
egrep -v 'SQL|pmon_timer|smon_timer|rdbms_ipc_message' znew01.txt > znew02.txt
join zold02.txt znew02.txt | sed 's/_/ /' | awk -f ora01.awk | sort -nr | awk -f ora02.awk | sed 's/_/ /g'
sleep 10
done
$ cat ora01.awk
{CentiSec = $4 - $3; if (CentiSec > 0) printf "%12d %8d %s\n", CentiSec, $1, $2}
$ cat ora02.awk
BEGIN {print "Session# Centiseconds Event\n-------- ------------ -----------------------------------"}
     {printf "%8d %12d %s\n", $2, $1, $3}
$
```

Using an endless loop, the script evtwaits provides statistic information in a new file znew01.txt, using a SQL command on the temporary table V$session_event. The new file is filtered to obtain only relevant information (SID, events and event_wait time) and a new file znew02.txt is generated. The files znew02.txt and zold02.txt are compared and the results are printed to the console, using .awk scripts, mentioned here later.

2. The results

2.1. First result: log file switch (*the "low-hanging fruit")*

Session#	Centiseconds	Event
→ **18**	**710**	**log file switch (checkpoint incomplete)**
15	502	db file scattered read
2	381	async disk IO

36	285	latch free
26	268	buffer busy waits
22	127	free buffer waits

From the very beginning we can locate the “low-hanging fruit” that will make this database perform better.

The alert log of this database will already have indicated to us via 'Checkpoint not complete' messages that there are not enough redo log groups to sustain the heavy workload associated with inserts, updates and deletes in the database.

The appropriate action [3] is therefore adding more redo log groups and doing so with special attention to disk placement.

2.2. I/O waits (*“db file scattered read”)*

Session#	Centiseconds	Event
18	710	log file switch (checkpoint incomplete)
→ **15**	**502**	**db file scattered read**
2	**381**	**async disk IO**
36	285	latch free
26	268	buffer busy waits
22	127	free buffer waits
.....		

Once the redo logs have been taken care of, what is going to show up as the worst offender event wait are disk I/Os, and from the name of the event - 'db file scattered read' - we can infer that these I/Os are caused by a full table scan - because scattered reads are multi-block reads, sequential ones are single-block reads.

If we can avoid full table scans by creating an additional index or influencing the optimizer [4], that may be the way to go, but keep in mind that some full table scans are legitimately performed because the alternative would have been indexed I/O that the optimizer deemed more expensive to perform because of the additional I/O against the index itself adding up past the point of diminishing return.

However, taking a holistic approach for our example, we suspect a cause-effect relationship between that full table scan I/O and the next event down,

'async disk IO'.

2.3. I/O Performance *("async disk IO")*

The 'async disk IO' wait problem may or may not go away once we size the log area properly.

In our particular configuration, the cause is a heavy insert activity against a table that is physically sitting on a raw VxVM volume.

If the full table scan, we saw before, needs to get blocks from the same physical disk(s) or not, we can see this in other ways by looking for high service times in output from **vxstat**, **iostat** commands [5] or the **SE** toolkit [9].

The appropriate action [4] is to isolate high activity tables in their own tablespaces, doing that with special attention to tablespace placement on disks.

2.4. Latch contention issues ("*latch free*")

Session#	Centiseconds	Event
18	710	log file switch (checkpoint incomplete)
15	502	db file scattered read
2	381	async disk IO
→ **36**	**285**	**latch free**
26	268	buffer busy waits
22	127	free buffer waits
.....		

If sizing the log area properly also eliminates the physical contention we saw, in the second and third entries, in our resource profile, our next focus needs to be on 'latch free' waits.

Waits on the 'latch free' event usually point to inefficient SQL statements that are repeatedly scanning the same buffer cache blocks over and over again, causing excessive CPU consumption as well as serialization on latches.

There are, however, other types of latches, but they do not present a problem in SAP R/3 databases running on Oracle 8.1.6 and above, because R/3 uses bind variables and no PL/SQL - minimizing the use of 'library cache' and 'shared pool' latches - and due to the way Oracle divides the buffer cache into a "cold" piece and a "hot" piece - by default 50/50 - significantly reducing the

number of times Oracle needs to acquire the 'cache buffers lru chain' latch to only those situations where a block migrates from being "cold" to being "hot".

A nested loops algorithm applied to joining very large tables visits the same blocks lots of times over and over again and will cause contention on the 'cache buffers chains' latch.

The solution is to apply the latest recommended Oracle patch if your research shows it was a bug with the CBO - or otherwise to get the optimizer to use a hash join algorithm instead, possibly using a combination of **stored outlines**, **inline views** and **/*+ USE_HASH */** hints [4].

2.5. Freelist contention issues (“*buffer busy waits*”)

Session#	Centiseconds	Event
18	710	log file switch (checkpoint incomplete)
15	502	db file scattered read
2	381	async disk IO
36	285	latch free
→ **26**	**268**	**buffer busy waits**
22	127	free buffer waits
.....		

Moving to the next step, 'buffer busy waits' may indicate one of several things, so we need more information from somewhere else.

The most frequent causes for such waits are:

a) not enough freelists on a table - causing serialization of inserts into that table by multiple sessions (i.e. R/3 work processes), and

b) not enough rollback segments - causing serialization of access to rollback segment header blocks when a session either needs to record the before image of a table row or otherwise during a block cloning operation leading to a "consistent get" from the buffer cache.

Multi-version read consistency is achieved by creating an identical copy of a buffer cache block in a new buffer slot and applying undo changes to the copy to make it look like a version of itself as of a previous point in time. The process of applying undo changes to a block clone requires serialized access not only to the undo block itself, but also to the header block of the corresponding rollback segment. There are far fewer undo header blocks than there are undo blocks, so

contention here will be a much bigger problem.

A simple query against V$WAITSTAT will reveal whether the Oracle instance has accumulated more 'buffer busy waits' belonging to the 'data block' class or to the 'undo header' class.

In the first case, the solution is to add freelists to the table. Adding freelists to a table can be done on-line [4, 3] and involves changing the table's STORAGE clause with an ALTER TABLE operation.

If the contention is on rollback segment header blocks the solution is to add more rollback segments [4, 3].

2.6. Free buffer waits

	Session#	Centiseconds	Event
	18	710	log file switch (checkpoint incomplete)
	15	502	db file scattered read
	2	381	async disk IO
	36	285	latch free
	26	268	buffer busy waits
→	**22**	**127**	**free buffer waits**
	.....		

The disk spindle contention we saw before has another negative side effect: the DBWR can't mark enough buffer slots as free to keep up with demand because it can't flush dirty blocks to disk fast enough.

Session ID 22 is trying to perform physical I/O to transfer blocks from disk into the buffer cache, but it can't find free buffer slots, so it has to wait for the DBWR process to provide them.

Few events we would hope not to find listed: 'free buffer waits' is one of them. If we succesfully performed previous actions, the DBWR works better and we will not see it again, when the output of the script has been updated.

3. The "trickle down" effect

In our example scenario, some actions taken (adding more redo log groups, implementing a better disk layout by isolating I/O intensive tables into their own physical storage, on-line adding of freelist to a table and applying the

latest recommended Oracle patch), provide not only relief for high I/O waits, but also indirectly for free buffer waits, because the DBWR will be able now to keep up more.

Generally speaking, this is true for other event waits as well. Removing the top bottleneck will not only make the top event wait disappear from the resource profile, but will also change event ranks and timings in the new profile.

Conclusions

Tuning by event waits is a repeatable step-by-step approach that finds out where the response time went every time a database bottleneck causes high response time sessions - which is what end-users care about;

The presented procedure:

a) is a *simple* script using facilities of operating system and database;
b) provides *on-line* relevant statistic information;
c) facilitates quantifiable *forecasts* and economically efficient decision making.

Being able to document ahead of time whether or not the target response time is going to meet the service level agreement will make it much easier to build a business case based on cost/benefit ratios and how-fast-on-ROI figures;

The presented procedure appeared as a technical extension of the economic theory of "*trickle down effect*" [8].

R E F E R E N C E S

1.*S.Iliescu s.a.* - Analiza de sistem în informatica industrială, Ed.Printech, Bucuresti 2000;
2.*G.Coulouris, J.Dollimore, T. Kindberg* - DISTRIBUTED SYSTEMS: Concepts and design, Addison-Wesley 2002;
3.*E. Bârlădeanu* – Analiza şi conducerea sistemelor informatice integrate de tip ERP. *Optimizarea performanţei*, Teză de doctorat, Universitatea "POLITEHNICA" Bucureşti, 2005;
4. *R. Niemec* - ORACLE Performance Tuning – Osborne/McGraw-Hill 2000;
5. *A. Cockroft* - Sun Performance and Tuning - 3rd Edition. O'Reilly & Associates, 2001;
6.*B. Rudiger* - Die Technologie des SAP - Systems: Basis fuer betriebwirtschaftliche Anwendungen, Addison – Wesley Longman 1998;
7. http://ftp.support.veritas.com/pub/support/products/;
8. http://www.investopedia.com/terms/t/trickledowntheory.asp
9. http://www.setoolkit.com
10. http://www.symmetrix.ch/