ORACLE AUTOMATIC SGA RESIZING: *WHY IT MAY NOT BE RIGHT FOR YOU*

by Chris Lawson

The Oracle database has become very sophisticated in recent years, as more and more new features are added. These new features, however, are occasionally accompanied by unanticipated (and unwanted) side effects. Here's a story of one new 10g feature run amok. For purposes of confidentiality, I will refer to the company as *Greg's Golf Supplies.*

THE SYMPTOMS

For unknown reasons, users at Greg's Golf would see application response time unpredictably increase 10-fold, from about ½ second to 5 seconds. This usually happened during busy times, but not always. We were able to detect this response blip because the company had sophisticated response time tools that kept track of the *longest* response time for any user during a 5-minute period. So, even though the averages were fine, the firm's monitoring tools detected these rare response time spikes. These occasional spikes really *were* important for business/legal reasons. (Remember, the company in this study isn't *really* a golf store.)

What made this problem especially hard to troubleshoot was its rare appearance. Compared to the huge volume of transactions in the database, these spikes were rare—only about 1% of all transactions. Thus, the cumulative runtimes and other statistics, as reported by AWR reports looked fine.

Since we had already detected a few improper queries in the application (e.g., scanning more data than necessary), we tended to assume that the application was the source of the response time blips.

LET THE GUESSING BEGIN!

A few analysts began focusing on the application servers, believing them to be the cause of the mischief. I pointed out that that could indeed be the case, but without actual evidence pointing that direction, it was just a guess. (I am always reluctant to go down paths based on hunches not supported by empirical facts.)

ASH REPORTS TO THE RESCUE

Using ASH (Active Session History), I was able to capture the "bad" sql, but it looked the same as the other "good" sql. In fact the "bad" sql was a very trivial query—simply getting the next value for a sequence.

Belatedly, I realized that the ASH views had vital information that I was overlooking. I should be looking more carefully at the wait event for the long-running sql. Using the view, DBA_HIST_ACTIVE_SESS_HISTORY, I summarized information on queries that took more than a few seconds to complete. Here is an example:

```
SAMPLE_TIME               SQL_TEXT                                        EVENT           CT
------------------------- ----------------------------------------------- --------------- ---
10-APR-08 05.18.AM select USER_HIST_seq.nextval from dual  library cache load lock      22
10-APR-08 06.17.AM select USER_HIST_seq.nextval from dual  library cache load lock      35
10-APR-08 06.21.AM select USER_HIST_seq.nextval from dual  library cache load lock      50
10-APR-08 07.32.AM select USER_HIST_seq.nextval from dual  library cache load lock      35
10-APR-08 09.44.AM select USER_HIST_seq.nextval from dual  library cache load lock      27
```

I spent hours analyzing the various sequences to no avail. Cache sizes were similar. Finally, I probed a bit deeper with my ASH scripts. I looked to see if the long-running sql were waiting on another session. Then, I checked what was obstructing the other session. Here's an example of one script I used to show the active sessions for a particular time:

```
With P1 As (Select Distinct Sample_Time, Event, Time_Waited,
Session_Id, Sql_Text, P1text,P1, P2text, P2,P3text,P3,Current_Obj#,
Current_File#,  Current_Block#, Blocking_Session
From Dba_Hist_Active_Sess_History A , V$Sqltext B
Where A.Sql_Id = B.Sql_Id
And Sample_Time Like '17-Apr-08 09.56.3%'
And Piece = 0
And Time_Waited/1000000 > 2)
Select Sample_Time, Blocking_Session, Event, Time_Waited/1000000 Sec, P1text,
P1, P2text, P2, P3text, P3
From P1
Order By Sample_Time;
```

A CONFUSING DISCOVERY

The results were confusing. I discovered that there was usually another session blocking my simple sql (the sequence next-value code), but this other session was waiting on this:

SGA: allocation forcing component growth

What in the world was that? Why should a session ever be waiting on SGA allocation? I examined SGA resizing activity by querying the view V$Sga_Resize_Ops:

```
Col When Format A25
Col Component Format A25
Select To_Char(Start_Time, 'Mon-Dd:Hh24:Mi:Ss') When ,
Component, Oper_Type, Initial_Size, Final_Size
From V$Sga_Resize_Ops;
```

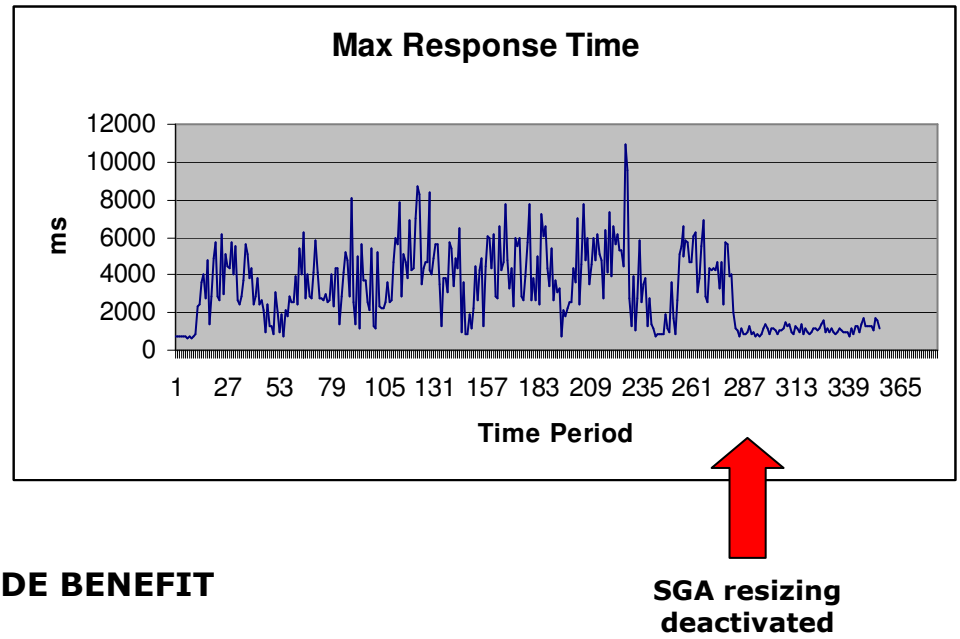
I was surprised by what I found out. It turns out that we were constantly changing the SGA up and down--sometimes many times per second. Here is an example of the "thrashing" we saw in production. In the sample below, Oracle did a GROW on the shared pool 3 times, and a SHRINK all within 1 second! This is obviously not how this feature is supposed to work.

WHEN	COMPONENT	OPER_TYPE	INITIAL_SIZE	FINAL_SIZE
APR-18:09:11:14	shared pool	SHRINK	939524096	905969664
APR-18:09:11:14	DEFAULT buffer cache	GROW	1.1979E+10	1.2012E+10
APR-18:09:11:14	DEFAULT buffer cache	SHRINK	1.2012E+10	1.1996E+10
APR-18:09:11:14	shared pool	GROW	905969664	922746880
APR-18:09:11:14	DEFAULT buffer cache	SHRINK	1.1996E+10	1.1979E+10
APR-18:09:11:14	shared pool	GROW	939524096	956301312

So Oracle was resizing the SGA at a furious rate—sometimes 10 times every second! I was quite sure this was not how this feature was supposed to work. I checked another production system and confirmed that the resizing should be infrequent. After a bit more investigation, I found other DBAs who experienced similar problems on high transaction systems. Additionally, a metalink note admitted that it may be wise to turn the feature off.

THE RESULTS ARE IN!

The production DBAs eagerly turned off the SGA automation by making a change in the init.ora file. We set *SGA_Target* = 0. With the automatic SGA feature restrained, the maximum application response time greatly improved—both in absolute value as well as consistency.



A NICE SIDE BENEFIT

After the fix was implemented, we also rid ourselves of the odd wait event, "cursor: pin S wait on X." Originally, this wait event was often in the top-5 waits in the AWR reports. In the excerpt below, for example, it is the third highest wait event.

AWR Top 5 Timed Events on busy morning

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
db file sequential read	3,781,176	24,158	6	46.4	User I/O
CPU time		14,578		28.0	
cursor: pin S wait on X	**381,951**	**6,465**	17	12.4	Concurrency
SQL*Net more data to dblink	848,934	3,298	4	6.3	Network
Streams capture: waiting for subscribers to catch up	2,308	2,567	1,112	4.9	Configuration

After the fix, the mystery wait event was no longer a major event. It was still there, but far, far down the list, with a total of just *2 seconds*!

CONCLUSION

Note that this performance improvement had absolutely nothing to do with bad application code. Unlike 98% of most performance fixes, the root cause really was the *database*—not the application. When was the last time you solved a performance problem by changing an init.ora parameter?

New features are usually helpful, but not always. It pays to be suspicious of hyped tools or algorithms that magically eliminate the need for DBA analysis.

Special credit to Jonathan Lewis for his work exposing the problems with the SGA automatic resizing feature. He has written several articles[1] on his experience with this feature.

Chris Lawson is an Oracle Performance Consultant and Oracle *Ace* who lives with his family in Dublin, California. He is the author of *Snappy Interviews: 100 Questions to Ask Oracle DBAs*, and *The Art & Science of Oracle Performance Tuning*, both available on Amazon.

Chris' web site is: www.OracleMagician.com.

In his spare time, Chris enjoys golf, choral singing (bass), and throwing frisbees to Morgan, the resident border collie.

[1] See, for instance, http://jonathanlewis.wordpress.com/2007/04/16/sga-resizing/