The White Papers

Resolving Oracle Latch Contention

By Guy Harrison

Principal Software Architect, Quest Software

Contents

Resolving Oracle Latch Contention

By Guy Harrison

Introduction

This white paper presents an overview of how the Oracle RDBMS uses latches to protect shared memory, the typical causes of and solutions to latch contention, and summarizes some research conducted at Quest Software that suggests that manipulating the (now) undocumented parameter "_spin_count" can be effective in relieving otherwise intractable latch contention problems.

What Are Latches?

Latches are serialization mechanisms that protect areas of Oracle's shared memory (the SGA). In simple terms, latches prevent two processes from simultaneously updating — and possibly corrupting — the same area of the SGA.

Oracle sessions need to update or read from the SGA for almost all database operations. For instance:

- When a session reads a block from disk, it must modify a free block in the buffer cache and adjust the buffer cache LRU chain[1].
- When a session reads a block from the SGA, it will modify the LRU chain.
- When a new SQL statement is parsed, it will be added to the library cache within the SGA.
- As modifications are made to blocks, entries are placed in the redo buffer.
- The database writer periodically writes buffers from the cache to disk (and must update their status from "dirty" to "clean").
- The redo log writer writes entries from the redo buffer to the redo logs.

Latches prevent any of these operations from colliding and possibly corrupting the SGA.

How Latches Work

Because the duration of operations against memory is very small (typically in the order of nanoseconds) and the frequency of latch requests very high, the latching mechanism needs to be very lightweight. On most systems, a single machine instruction called "test and set" is used to see if the latch is taken (by looking at a specific memory address) and if not, acquire it (by changing the value in the memory address).

If the latch is already in use, Oracle can assume that it will not be in use for long, so rather than go into a passive wait (e.g., relinquish the CPU and go to sleep) Oracle will retry the operation a number of times before giving up and going to passive wait. This algorithm is called acquiring a *spinlock* and the number of "spins" before sleeping is controlled by the Oracle initialization parameter "_spin_count".

The first time the session fails to acquire the latch by spinning, it will attempt to awaken after 10 milliseconds. Subsequent waits will increase in duration and in extreme circumstances may exceed one second. In a system suffering from intense contention for latches, these waits will have a severe impact on response time and throughput.

[1] The LRU (Least Recently Used) chain records how often individual blocks have been accessed. If a block is not accessed it moves towards the LRU end of the list and eventually is flushed from the cache.

Causes of contention for specific latches

The latches that most frequently affect performance are those protecting the buffer cache, areas of the shared pool and the redo buffer.

- **Library cache latches:** These latches protect the library cache in which sharable SQL is stored. In a well defined application there should be little or no contention for these latches, but in an application that uses literals instead of bind variables (for instance "WHERE surname='HARRISON'" rather that "WHERE surname=:surname," library cache contention is common.
- **Redo copy/redo allocation latches:** These latches protect the redo log buffer, which buffers entries made to the redo log. Recent improvements (from Oracle 7.3 onwards) have reduced the frequency and severity of contention for these latches.
- **Shared pool latches:** These latches are held when allocations or de-allocations of memory occur in the shared pool. Prior to Oracle 8.1.7, the most common cause of shared pool latch contention was an overly large shared pool and/or failure to make use of the reserved area of the shared pool[2].
- **Cache buffers chain latches:** These latches are held when sessions read or write to buffers in the buffer cache. In Oracle8i, there are typically a very large number of these latches each of which protects only a handful of blocks. Contention on these latches is typically caused by concurrent access to a very "hot" block and the most common type of such a hot block is an index root or branch block (since any index based query must access the root block).

Measuring Latch Contention

Ratio-based Techniques

Conventional wisdom in the mid-90s was to focus on the latch "miss" rate to determine the degree of latch contention. Looking at the view v$latch:

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
ADDR                                               RAW(4)
LATCH#                                             NUMBER
LEVEL#                                             NUMBER
NAME                                               VARCHAR2(64)
GETS                                               NUMBER
MISSES                                             NUMBER
SLEEPS                                             NUMBER
IMMEDIATE_GETS                                     NUMBER
IMMEDIATE_MISSES                                   NUMBER
WAITERS_WOKEN                                      NUMBER
WAITS_HOLDING_LATCH                                NUMBER
SPIN_GETS                                          NUMBER
SLEEP1                                             NUMBER
SLEEP2                                             NUMBER
SLEEP3                                             NUMBER
SLEEP4                                             NUMBER
SLEEP5                                             NUMBER
SLEEP6                                             NUMBER
SLEEP7                                             NUMBER
SLEEP8                                             NUMBER
SLEEP9                                             NUMBER
SLEEP10                                            NUMBER
SLEEP11                                            NUMBER
```

[2] The reserved area of the shared pool is intended to hold large contiguous entries in the shared pool (such as large PL/SQL packages).

We see that for each latch, the number of gets (requests for the latch), misses (number of times the first request fails) and sleeps (number of times a session failed to obtain a latch by spinning) are recorded. In the past, queries such as the following were often used to determine latch health:

```
SQL> select name, gets, misses, misses*100/gets misspct from v$latch where gets>0;

NAME                                            GETS      MISSES MISSPCT
---------------------------------------- ----------- ----------- -------
latch wait list                                   32           1    3.13
process allocation                                28           0     .00
session allocation                         1,223,068          84     .01
session switching                              8,009           0     .00
process group creation                            40           0     .00
session idle bit                           2,426,940           1     .00
shared java pool                               1,188           0     .00
event group latch                                 28           0     .00
messages                                   2,128,851         461     .02
enqueues                                   3,168,279           7     .00
enqueue hash chains                        1,747,312           2     .00
channel handle pool latch                         40           0     .00
channel operations parent latch                  158           3    1.90
message pool operations parent latch               3           0     .00
file number translation table                    946           0     .00
mostly latch-free SCN                              1           0     .00
cache buffers lru chain                      343,827           0     .00
active checkpoint queue latch                174,293           0     .00
checkpoint queue latch                     1,459,929          26     .00
cache buffers chains                      13,851,179      16,254     .12
cache buffer handles                              61           0     .00
```

This approach was flawed on a number of levels:

- It is actually the number of sleeps that most accurately influences the impact of the latch contention on response time.
- A high miss rate is expected for certain latches.
- A latch with a high miss rate (or sleep rate) that is not frequently accessed is probably not impacting performance.
- Even if a latch is experiencing a high sleep rate, we can't determine the impact on performance without taking into account waits for other resources. So if sessions are waiting 90% for IO, 8% for CPU and 2% for latch, expending effort on halving the latch sleep wait only provides a 1% improvement in response time – probably not noticeable.

In the above example the "latch wait list" latch has the highest miss rate. However, this is totally irrelevant since it was only requested 26 times, while the "cache buffer chains" latch appears to have only a moderate miss rate, but has been requested almost three million times and — as we shall see — is the latch most affecting performance.

Wait interface-based techniques

A better approach to estimating the impact of latch contention is to consider the relative amount of time being spent waiting for latches. The following query gives us some indication of this:

```
SELECT   event, time_waited,
         round(time_waited*100/ SUM (time_waited) OVER(),2) wait_pct
    FROM (SELECT event, time_waited
            FROM v$system_event
           WHERE event NOT IN
                    ('Null event',
                     'client message',
                     'rdbms ipc reply',
                     'smon timer',
                     'rdbms ipc message',
                     'PX Idle Wait',
                     'PL/SQL lock timer',
                     'file open',
                     'pmon timer',
                     'WMON goes to sleep',
                     'virtual circuit status',
                     'dispatcher timer',
                     'SQL*Net message from client',
                     'parallel query dequeue wait',
                     'pipe get'
                    ) UNION
         (SELECT NAME, VALUE
            FROM v$sysstat
           WHERE NAME LIKE 'CPU used when call started'))
ORDER BY 2 DESC

EVENT                          TIME_WAITED   WAIT_PCT
------------------------------ ----------- ----------
latch free                           40144      31.67
CPU used when call started           30341      23.94
control file sequential read         12341       9.74
direct path read                     11933       9.41
control file parallel write           6487       5.12
file identify                         5666       4.47
log file sync                         3492       2.75
log file parallel write               3213       2.53
instance state change                 3064       2.42
log file switch completion            3049       2.41
db file sequential read               2290       1.81
```

Now we can look at the sleeps in v$latch to determine which latches are likely to be contributing most to this problem:

```
SQL> select name, gets, sleeps,
            sleeps*100/sum(sleeps) over() sleep_pct, sleeps*100/gets sleep_rate
       from v$latch where gets>0
       order by sleeps desc;

NAME                                  GETS       SLEEPS SLEEP_PCT SLEEP_RATE
------------------------------ ----------- ------------ --------- ----------
cache buffers chains            13,863,552       38,071     99.48      .2746
session allocation               1,223,982          110       .29      .0090
checkpoint queue latch           1,461,039           39       .10      .0027
library cache                    9,239,751           22       .06      .0002
shared pool                        869,652           16       .04      .0018
messages                         2,130,515            6       .02      .0003
redo writing                     1,330,987            6       .02      .0005
latch wait list                         33            0       .00      .0000
session switching                    8,014            0       .00      .0000
session idle bit                 2,428,851            0       .00      .0000
enqueues                         3,171,018            0       .00      .0000
channel handle pool latch               40            0       .00      .0000
message pool operations parent           3            0       .00      .0000
 latch
mostly latch-free SCN                    1            0       .00      .0000
```

Now we are in a position to make some reasonable conclusions:

- Latch sleeps contribute to about 30% of database response time (very excessive), AND
- It's the cache buffers chains latches that contributes to the vast majority of these waits.

Note that if we had used the conventional "ratio based" analysis outlined in the previous section we would have discounted cache buffers chains latches as a problem because the miss rate was "only" 0.15%.

Tuning the Application to Avoid Latch Contention

There are some things we can do within our application design that can reduce contention for latches.

Using Bind Variables

As noted earlier, failure to use bind variables within an application is the major cause of library cache latch contention. All Oracle applications should make use of bind variables whenever possible.

However, all is not lost if you are unable to modify your application code. From 8.1.6 onwards you can use the "CURSOR_SHARING" parameter to cause Oracle to modify SQL on the fly to use bind variables. A setting of FORCE causes all literals to be converted to bind variables. A setting of SIMILAR causes statements to be rewritten only if it would not cause the statements execution plan today (which can happen if there are histogram statistics defined on a column referenced in the WHERE clause).

Avoiding Hot Blocks

Cache buffers chains latch contention is one of the most intractable types of latch contention. There are a couple of things you can do at the application level to reduce the severity of this type of contention.

Firstly, identify the blocks that are "hot." Metalink note 163424.1, "How to Identify a Hot Block Within The Database" describes how to do this. Having identified the identity of the hot block, you will most likely find that it is an index root or branch block. If this is the case, there are two application design changes that may help.

1) Consider partitioning the table and using local indexes. This might allow you to spread the heat amongst multiple indexes (you will probably want to use a hash partition to ensure an even spread of load amongst the partitions).
2) Consider converting the table to a hash cluster keyed on the columns of the index. This allows the index to be bypassed completely and may also result in some other performance improvements. However, hash clusters are suitable only for tables of relatively static size, and determining an optimal setting for the SIZE and HASHKEYS storage parameters are essential.

Is Latch Contention Inevitable?

While conducting performance tuning consultancies or visiting customer sites over the years I have noticed that the most highly optimized databases running on the most high end hardware seem to be the ones that suffer most significantly from latch contention.

It would appear that as we remove all other constraints on database performance, contention for latches becomes the ultimate limiting factor on database throughput.

Imagine we have a perfectly tuned application: we have allocated sufficient memory to the SGA and have a sufficiently low latency IO sub-system that waits for IO are negligible. CPU is abundant and exceeds the demands of the application. When we reach this desirable state, the database will be doing almost nothing but performing shared memory accesses and hence, latches – which prevent simultaneous access to the same shared memory areas. This will become the limiting factor.

Some degree of latch contention – probably on the cache buffers chains latch – has to be accepted in very high volume systems with extremely powerful hardware support.

When all other constraints on the database are removed, contention for shared memory will probably be the limiting factor on performance.

Investigating _spin_count

Prior to Oracle 8.1, the spin count parameter (_spin_count or latch_spin_count) was a documented parameter and many DBAs attempted to adjust it to resolve latch contention. However, as of Oracle8i the parameter is "undocumented" (e.g., does not appear in v$parameter and is not documented in the Oracle reference manual). Why did Oracle do this? The official Oracle Corporate line is that the value of _spin_count is correct for almost all systems and that adjusting it can cause degraded performance. For instance, Metalink Note:30832.1 says: "If a system is not tight on CPU resource _spin_count can be left at higher values but anything above 2000 is unlikely to be of any benefit."

However, I believe that higher values of _spin_count can relieve latch contention in many circumstances and I think Oracle depreciated the parameter incorrectly.

In this section I will present the results of some experiments I conducted into the effect of adjusting spin count on the performance of a system suffering from heavy latch contention. In my research, I created a simulation in which cache buffers chains latch contention became very severe as a result of hot index blocks. I then adjusted _spin_count programmatically across a wide range of values and recorded the impact on database throughput, latch waits and CPU utilization.

Relationship Between _spin_count, Latch Contention and Throughput

Figure 1 summarizes the relationship between database throughput (as measured by the number of SQL statement executions per second), the amount of time spent in 'latch free' passive waits and the CPU utilization of the system.

The data indicates that as _spin_count increased, waits for latches reduced as CPU utilization increased. As CPU utilization approached 100%, improvements in throughput ceased even while latch free wait time continued to reduce.

If throughput fails to increase even as latch free waits decrease then logically, Oracle sessions must be waiting for another resource. The obvious suspect resource is the CPU. Figure 2 illustrates how throughput relates to the CPU run queue (the number of processes waiting for the CPU at any time). As we can see, throughput starts to drop off as the run queue approaches a value of one (which means in effect that sessions will usually wait for the CPU when they attempt to spin and when they awake after a sleep).

Note that the optimal value for _spin_count in this simulation was somewhere in the vicinity of 12,000 — six times the default value provided by Oracle and that throughput more than doubled at this point.

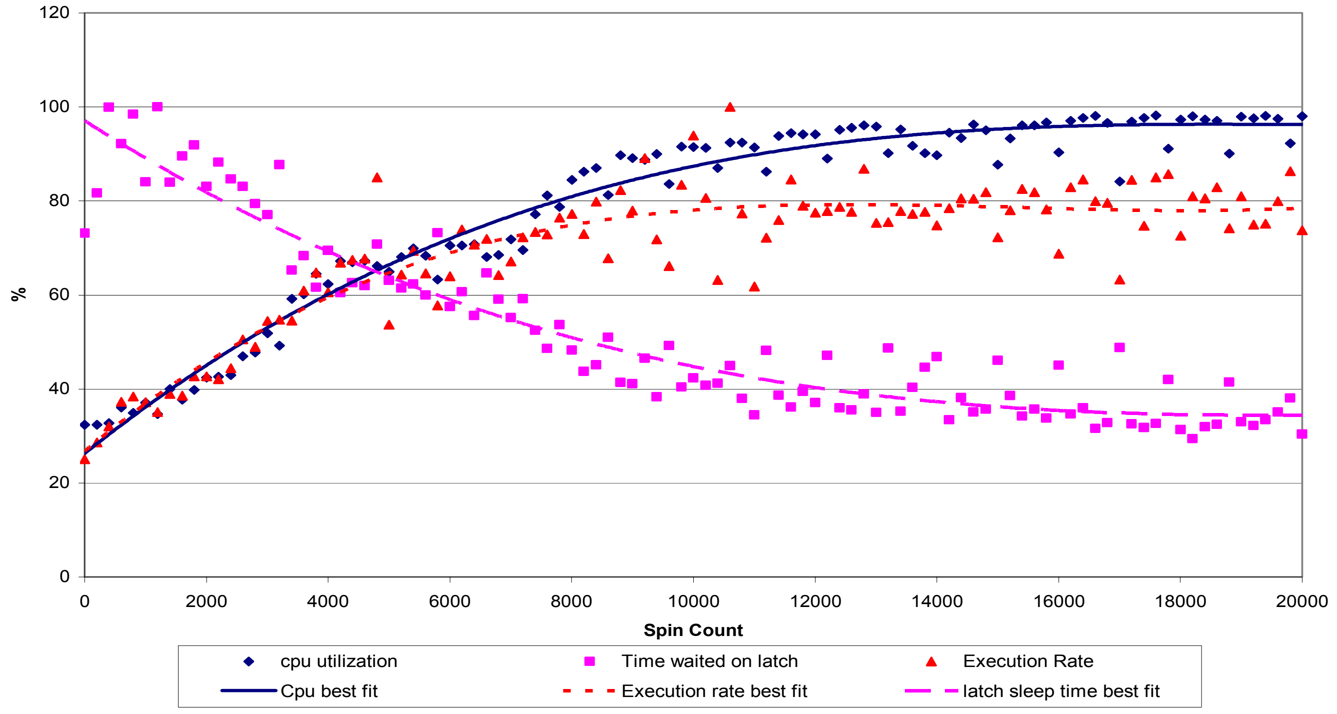


Figure 1: Relationship between _spin_count, CPU utilization, throughput and latch wait

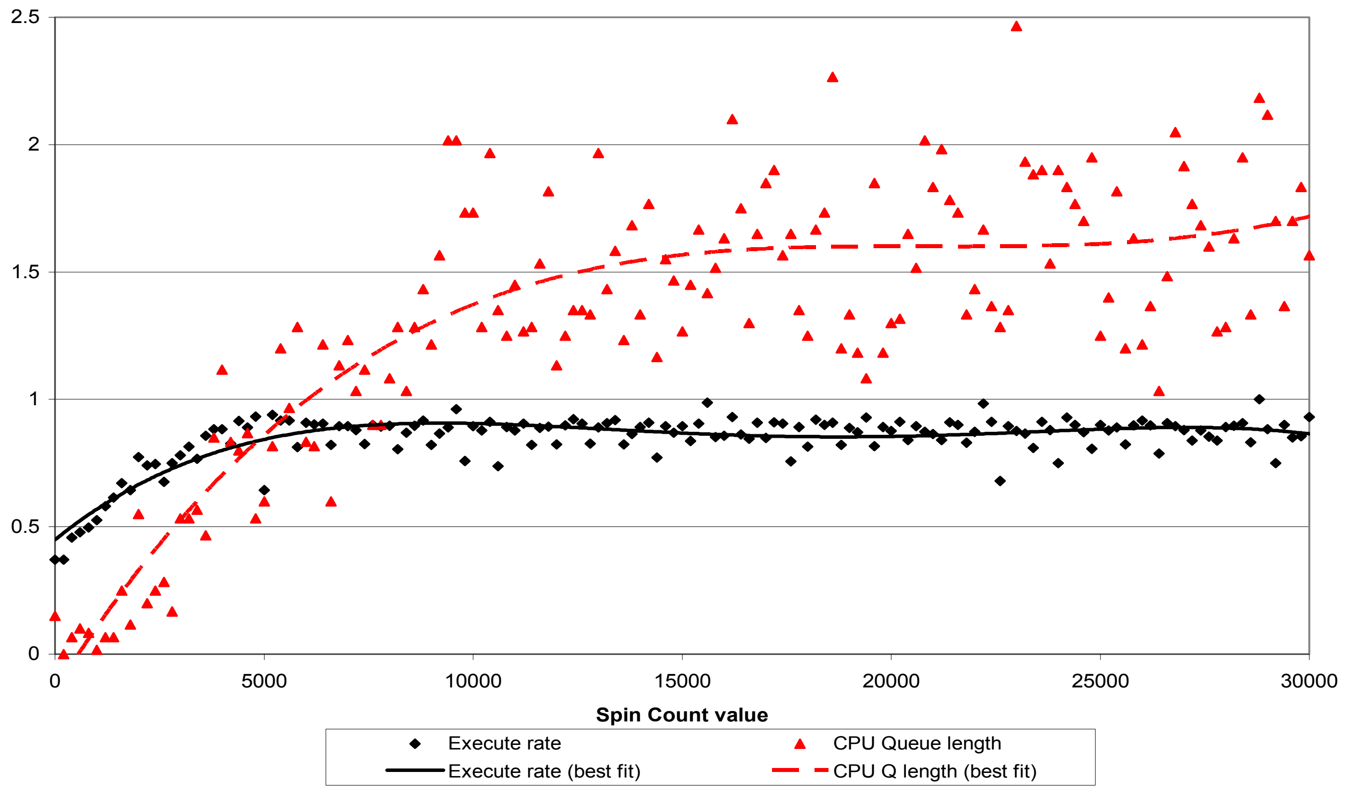


Figure 2: Relationship between _spin_count, CPU queue length and database throughput

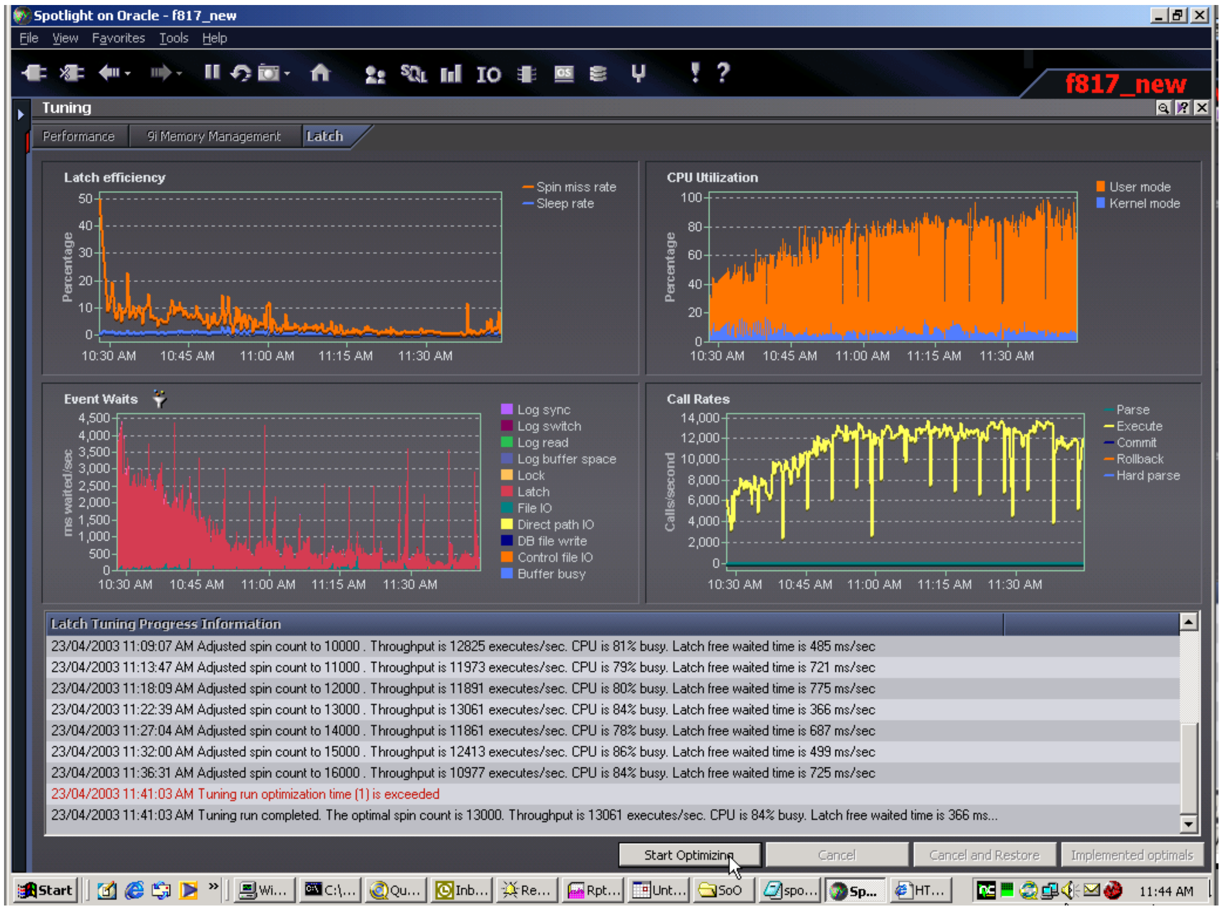
Conclusion

We've seen clearly that at least in some circumstances, manipulating the value of _spin_count can result in very significant reductions in latch free waits and improve the throughput of latch-constrained applications. Unfortunately, as an undocumented parameter, some DBAs will be reluctant to manipulate _spin_count. However, if faced with intractable latch contention — particularly for cache buffers chains latches — manipulating _spin_count may be the only available option to improve database throughput.

_Spin_count should only be adjusted when there are available CPU resources on the system. Specifically, if the average CPU Queue length is approaching or greater than 1, then increasing _spin_count is unlikely to be effective.

Oracle set the default value of _spin_count to 2000 in Oracle7. Over the subsequent 10 or so years, CPU clock speed has increased by more than a factor of 10. This means that Oracle systems are spending a decreasing amount of time spinning while the latch sleep time has remained pretty much constant. So it is arguable that the default value of _spin_count should have been increased with each release of Oracle.

Tuning _spin_count requires that you carefully monitor CPU utilization and throughput to determine if the change has been effective. At Quest, we have implemented an automatic facility that does this within Quest Central. The latch-tuning module is shown in Figure 3.



```
23/04/2003 11:09:07 AM Adjusted spin count to 10000 . Throughput is 12825 executes/sec. CPU is 81% busy. Latch free waited time is 485 ms/sec
23/04/2003 11:13:47 AM Adjusted spin count to 11000 . Throughput is 11973 executes/sec. CPU is 79% busy. Latch free waited time is 721 ms/sec
23/04/2003 11:18:09 AM Adjusted spin count to 12000 . Throughput is 11891 executes/sec. CPU is 80% busy. Latch free waited time is 775 ms/sec
23/04/2003 11:22:39 AM Adjusted spin count to 13000 . Throughput is 13061 executes/sec. CPU is 84% busy. Latch free waited time is 366 ms/sec
23/04/2003 11:27:04 AM Adjusted spin count to 14000 . Throughput is 11861 executes/sec. CPU is 78% busy. Latch free waited time is 687 ms/sec
23/04/2003 11:32:00 AM Adjusted spin count to 15000 . Throughput is 12413 executes/sec. CPU is 86% busy. Latch free waited time is 499 ms/sec
23/04/2003 11:36:31 AM Adjusted spin count to 16000 . Throughput is 10977 executes/sec. CPU is 84% busy. Latch free waited time is 725 ms/sec
23/04/2003 11:41:03 AM Tuning run optimization time (1) is exceeded
23/04/2003 11:41:03 AM Tuning run completed. The optimal spin count is 13000. Throughput is 13061 executes/sec. CPU is 84% busy. Latch free waited time is 366 ms...
```

Figure 3: Quest Central's automatic latch tuning module.

About the Author

Guy Harrison is a principal software architect for Quest Software and a recognized Oracle expert. With more than 15 years experience in Oracle administration, development, performance tuning and project management, Guy is the author of Oracle SQL High Performance Tuning (Prentice-Hall, 1997,2000) and Oracle Desk Reference (Prentice-Hall, 2000) and is a regular contributor to Oracle-related technical journals. Guy was instrumental in the creation of Spotlight®, a popular diagnostic tool as well as contributing to the development of other Quest products such as Quest Central™.

About Quest Software

Quest Software, Inc. (NASDAQ: QSFT) is a leading provider of application management solutions. Quest provides customers with Application Confidence^{sm} by delivering reliable software products to develop, deploy, manage and maintain enterprise applications without expensive downtime or business interruption. Targeting high availability, monitoring, database management and Microsoft infrastructure management, Quest products increase the performance and uptime of business-critical applications and enable IT professionals to achieve more with fewer resources. Headquartered in Irvine, Calif., Quest Software has offices around the globe and more than 18,000 global customers, including 75% of the Fortune 500. For more information on Quest Software, visit www.quest.com.

World Headquarters
8001 Irvine Center Drive
Irvine, CA 92618
www.quest.com
e-mail: info@quest.com
Inside U.S.: 1.800.306.9329
Outside U.S.: 1.949.754.8000

Please refer to our Web site for regional and international office information. For more information on Quest Central, please visit http://www.quest.com/quest_central/qco/.