The optimal way to get more from your existing database system investment, and how bad performance can affect your business

Joakim Treugut

Chief Oracle Investigator

Synergy International Ltd

Abstract

Many systems' resources are used in a non-optimal way. You don't think you have performance problems, but do you know the maximum speed of your system? Do you still use buffer cache hit ratio as a measurement for a system's performance?

Buying more hardware and silicon when the existing system "feels too small" cannot replace the performance improvements you can get by optimising the workload. At best, a second CPU can give twice as much throughput, whereas tuning the application, SQL and the Oracle instance can give 10-1000 times more than that.

This workshop is about queueing theory in practice, and how to get more out of existing investments and systems. By making service time more efficient, we reduce the wait time, which is a win-win for the response time, your business and your investment.

Introduction

This workshop will not be so much about Oracle and what is new in 9i. I will talk about a way to think about and work with performance problems. This can be applicable to any database, but I will use Oracle terms and what information Oracle can give us when we are handling a performance problem. I will start to talk about how bad performance can affect a company's business. Then I will continue with three ways to optimise a database system: hit ratios, wait events and SQL statements' response time. But before I start to talk about the third method, I will give an introduction about queueing theory.

What do I want you to remember from my presentation:

- Bad performance has a negative impact on your business (losing customers or investing in more hardware) and reputation. Time is money. Losing time is losing money.
- DBAs, don't use buffer cache hit ratio or other hit ratios as a basis for your tuning!
- Managers, don't accept performance reports that mention and correlate high instance hit ratios with good performance!
- If the problem is bad performance, then buying more and faster hardware, is only the solution when the root cause is lack of system resources, otherwise it is a fix. A fix can hide the symptoms, and delay its return. If the problem is contention on a serial resource (redo allocation latch, shared pool latch) then a hardware upgrade will not solve the problem.
- Low buffer cache hit ratio on session and statement level can be used to find suspicious sessions/statements that are doing a lot of physical reads.
- A high buffer cache hit ratio does not mean that your database is optimally tuned, and it is often a good signal that your system has a lot of SQL that is doing inefficient logical reads.
- Summarised wait event statistics (from v$system_event) can be good to use as a starting point for finding bottlenecks and optimising the instance, but should not be used alone to implement changes.

- Don’t start by trying to eliminate queues and wait time. There is a reason why there is a queue. Start by looking at what is going on at the counter and the service time that is spent there. By decreasing the queue, only the wait time will decrease. By decreasing the service time, the wait time will also be decreased.

- The biggest benefit often comes from looking at the SQL statements, their response time and execution plans (to find suspicious SQL use v$sql together with v$sql_plan_statistics_all, or use the trace event 10046 with level 8).

- Developers, optimise the code while you write it!

How bad performance can hurt a company’s business and reputation

No one likes to wait; the time is wasted, and time is money – wasted money.

A lot of money is spent on implementing and maintaining your system. You have invested in hardware, 3rd party applications, licenses, development, maintenance and support. You want to maximise your investment and get good return from it. But a chain is not stronger than the weakest link.

You are driving to your summer house as you have done every Friday after work for ten years; the trip takes two hours. Then one Friday there is a big queue, so the trip takes two and a half hours. The Friday after that it is the same. If you can take another way so the trip still takes around two hours, how many Fridays will you give your old road a chance until you change?

A system can be good and its users can have the perception that the system is good. If the system becomes bad, the users can still have the perception of a good system. In this situation, if the company takes the actions that are required and makes the system good again, then the users will never notice this change. But if the company doesn’t do anything, then the users will get the perception that this system is bad. If the users abandon the system, it will be difficult for the company to get them back. It is not enough for the company to make the system good. They must take other actions (marketing, advertising…) so the users’ perception of a good system is returned.

What consequences does it have for you and your business when the most important application is not available, takes twice as long time, or is terminated with an error?

	The quality of the system is good	**The quality of the system is bad**
The users’ perception is that the system’s quality is good	1. A company has a good system (availability, performance, security, easy to use), and its users think it is a good system.	2. For some reason the quality of the system has changed, but the users have not noticed it yet. Maybe they have nothing to compare it with. If the company brings the system back to good quality, then the users will not notice anything.
The users’ perception is that the system’s quality is bad	4. If the users are lost, it is difficult for the company to bring them back, even if the company has improved the quality of the system. The users have found another good system to use, so why should they change. To get the users back, the company needs to advertise and the users need to get some bad experiences with the system they are using.	3. But if the users get the perception that the quality is bad, or that they have better choices, then the users may start to abandon the system at this point.

Three different ways to optimise a database system

There are many different ways to monitor and optimise a database system. One is to use hit and miss ratios (buffer cache hit ratio, library cache hit ratio, data dictionary hit ratio, latch miss ratio, disk sorts ratio…). Another is to look at the summarised statistics for wait events that Oracle collects in v$system_event, sort the output after highest wait time and start with the top non-idle event – to remove the biggest bottleneck. A third way is to concentrate on what is controlling the activity in a database system: the SQL statements. An instance without SQL statements would be infinitely fast (no queues exist, no-one is waiting and response time would be zero).

Good buffer cache hit ratio doesn't exist – only high

It should be easy to prove mathematically that buffer cache hit ratio cannot be used to determine the performance of an Oracle instance. If we only could find a measurable definition for what "good performance" is. Then we could use the correlation coefficient between buffer cache hit ratio and performance, and see what result we get. The closer to 1.0 that we get, the more we can rely on the correlation between them. But I have seen Oracle systems with low buffer cache hit ratio and happy users and upset users. And I have seen Oracle systems with high buffer cache hit ratio and happy users and upset users. So my observations are far away from the regression line.

I investigated a system that had a buffer cache hit ratio around 70%, and the DBA told me that she had increased the buffer cache several times, but could not see any better hit ratio. I asked if she had any complaining users, and she said that all of them were very happy about the performance. The reason was that 180 sessions (the users) had a session buffer cache hit ratio around 99%. But a daily batch job had 56% in session buffer cache hit ratio. For that session, most statements had 50-100% in statement buffer cache hit ratio, and one statement had 3%, and was reading hundred of thousands of blocks from disk, over and over again. A bigger buffer cache didn't help this session due to the way Oracle cache blocks from a full-table scan of a "large" table, and a bigger buffer cache didn't help the 180 sessions that were reading from totally different tables, which were already cached.

Use v$sess_io and v$sql (if you want to make me happy, avoid using v$sqlarea, it mixes apples with pears, and is doing a horrible group by on the text of the statement) to find the buffer cache hit ratio on session and statement level.

The buffer cache hit ratio on an instance level doesn't say anything about who did these physical reads, when they occurred, why they occurred, was anyone affected…

Who did all these physical reads? Was it one session or many? Did they come from one statement or many? A monitor tool that every minute executes SELECT * FROM v$datafile, will produce <number of datafiles> * 2 physical reads every minute (a way to reduce that down to zero is if "SELECT *" can be replaced with "SELECT file#, name"). If one session is reading a few tables with a lot of physical reads and experiences bad performance, and 199 other sessions are reading other tables, mostly with logical reads, they will not recognise the bad performance.

When did the physical reads occur? Was it during daytime or was it a big nightly batch job that is refreshing materialized views, analysing tables and indexes, and taking a full export? If the night job is the only job running, no one is affected if the job takes one or four hours to complete. If the night job does a lot of physical reads, so the instance buffer cache hit ratio is low, the day time users can still experience splendid performance.

Why did the physical reads occur? Of course they occurred because they were not already in the buffer cache. But why were they not in the buffer cache? Was it the first time the block was read since startup? Is the block read by a full table scan and does it belong to a table that is larger than _small_table_threshold so it is flushed out immediately? Is it because we are analysing all the segments for a schema with the compute option or taking a full export? Are we reading something that can never be cached (reading from v$datafile_header, or are we using parallel execution which does physical reads that by-passes the buffer cache)? Has the block been in the buffer cache before and now does it need to be read back, because the buffer cache is too small?

Was anyone affected and how much? When Oracle reports that a physical read was done, Oracle has no idea if the block was really read from disk or already existed in the UNIX filesystem buffer cache, this affects the wait time for the read, but is still counted as a physical read from Oracle's point of view. We cannot see from the buffer cache hit ratio if the users were affected and how much? So in Oracle7, the wait interface was introduced.

How you can get a higher buffer cache hit ratio

Because the formula for buffer cache hit ratio is

1 – (physical reads that have passed the buffer cache / logical reads from the buffer cache)

there are two ways to get a higher (remember that higher is not the same as better) buffer cache hit ratio. One is to *decrease physical reads* (which is good). The other one is to *increase logical reads* (which is bad). An instance with a very high buffer cache hit ratio has often a lot of SQL statements that repeatedly do unneeded logical reads.

If only hit ratios are used, a query like the one below (executed in a 8.1.7 instance) will never be found. By executing this query many times, the buffer cache hit ratio will increase for the whole instance.

```
SELECT level, mgr, empno, ename
  FROM big_emp2
 START WITH  mgr IS NULL
CONNECT BY mgr = PRIOR empno;
```

Without index on MGR (index on EMPNO doesn't matter):

```
real: 254255 (254.255 seconds)

Execution Plan
0    SELECT STATEMENT Optimizer=CHOOSE (Cost=40 Card=1575 Bytes=61425)
1  0   CONNECT BY
2  1     TABLE ACCESS (FULL) OF 'BIG_EMP2' (Cost=40 Card=1 Bytes=13)
3  1     TABLE ACCESS (BY USER ROWID) OF 'BIG_EMP2'
4  1     TABLE ACCESS (FULL) OF 'BIG_EMP2' (Cost=40 Card=1575 Bytes=61425)

Statistics
   56688  db block gets         => This will give a buffer
 2456460  consistent gets       => cache hit ratio on 100%
       0  physical reads
       7  sorts (memory)
       0  sorts (disk)
    9447  rows processed
```

With an index on MGR (index on EMPNO doesn't matter):

```
real: 12749 (12.749 seconds)

Execution Plan
0     SELECT STATEMENT Optimizer=CHOOSE (Cost=14 Card=1575 Bytes=61425)
1  0    CONNECT BY
2  1      TABLE ACCESS (FULL) OF 'BIG_EMP2' (Cost=40 Card=1 Bytes=13)
3  1      TABLE ACCESS (BY USER ROWID) OF 'BIG_EMP2'
4  1      TABLE ACCESS (BY INDEX ROWID) OF 'BIG_EMP2' (Cost=14 Card=1575)
5  4        INDEX (RANGE SCAN) OF 'IX_BIG_EMP2_MGR' (NON-UNIQUE) (Cost=4)
```

```
Statistics
     6  db block gets
 29230  consistent gets
     0  physical reads
     1  sorts (memory)
     0  sorts (disk)
  9447  rows processed
```

These queries were executed on a server with only one user. In a multi-user system, the first query would steal 241 (254-13) CPU seconds from other users, which will let them wait in the CPU's run-queue, but produce "nice" instance buffer cache hit ratio. Is this good performance?

In Oracle9i there is a new operation called CONNECTION BY PUMP, which gives the same good response time as the second query even if no index exists on the MGR column.

References:

Good buffer cache hit ratio doesn't exist! Why high buffer cache hit ratio is useless to tell the performance of an Oracle instance, Joakim Treugut, Synergy International Limited (available on request)

Removing bottlenecks by using summarised history statistics – Oracle's wait interface

In Oracle7 release 7.0.12 the wait interface was introduced. Today in 9.2.0 there are 361 different wait events.

When a session needs to wait, it records in v$session_wait what event it is waiting for (seq# is increased by 1 and wait_time is set to 0). When the wait is over wait_time, v$session_event and v$system_event are updated.

If timed_statistics is not set, then these views are not so useful (time_waited is not updated, only total_waits and total_timeouts). It doesn't help me that the sessions have been waiting 1000 times on latch free, when it doesn't say for how long.

The second popular approach to optimising a database system is to get the rows from v$system_event sorted by time_waited and remove the biggest bottleneck. This reason is much better than focusing on hit ratios, but it is still not perfect. The reasons why, are 1. It only focuses on wait time (to be correct, an Oracle wait event contains some system/kernel service time so it is more a response time), not service time. 2. The waits are often symptoms and not the real cause. We see high wait time on "latch free" and increase spin_count (now an hidden/underscore parameter) and introduce new problems or move the bottleneck somewhere else. By fixing the symptoms we hide the real problems for some time, but they will return. 3. It can only be used to address problems with the configuration of the Oracle instance; it cannot be used to solve problems with the data design, application design or the code.

```
EVENT                          TOTAL_WAITS TIME_WAITED AVERAGE_WAIT
rdbms ipc message                    15045     6045874          402
pmon timer                            4317     1268128          294
smon timer                              46     1236255        26875
db file scattered read             2659903      922925            0
SQL*Net message from client            235      388074         1651
control file sequential read          1764        3979            2
db file sequential read               1338        3209            2
enqueue                                 10        3090          309
log file parallel write               1221        1225            1
db file parallel write                 987         156            0
```

This method is a little better to give answers to the questions “when did these waits occur”, “why did they occur” and “was anyone affected”. We know that a lot of these waits occur because one session is waiting for another, so someone was affected. We can see the wait time, but we can’t measure the pain. If we are lucky we can still find the session that is responsible for the high waits in v$session_event. If we have high wait time for *enqueue*, we know that one session wants a lock that someone else holds; we can find what type of lock (from v$enqueue_stat or x$ksqst), but not on which resource. But we have a good starting point; that we shall monitor v$lock to get more information. So don’t use this information to draw your conclusions and recommend changes, but use this information as an excellent starting point for where to look next. In Oracle 9.2 you get the detailed information (in v$segment_statistics) that you need to narrow down the problem.

If we see waits on…	**Then we can find information here…**	**And even more here for Oracle 9i**
buffer busy waits	v$waitstat, x$kcbfwait, x$kcbwh, x$kcbsw	v$segment_statistics
db file scattered read	v$filestat, v$sess_io, v$sql,	v$segment_statistics, v$sql_plan_statistics_all
db file sequential read	v$filestat , v$sess_io, v$sql	v$segment_statistics, v$sql_plan_statistics_all
Enqueue	v$lock, x$ksqst	v$segment_statistics, v$enqueue_stat
latch free	v$latch, v$latch_parent, v$latch_children, v$latchholder	v$latch.wait_time
library cache lock	x$kgllk	
library cache pin	x$kglpn	

Throwing silicon at the symptoms

It is easy to introduce new problems and hide the real cause by only monitoring hit ratios or wait events, and use them as a basis for conclusions and recommendations. For example:

- The disk-sort ratio is above 5% so I increase sort_area_size to make more memory sorts possible, but then some of my queries are not doing efficient nested loops anymore. The higher value of sort_area_size made sort/merge joins and hash joins cheaper, so now the system is doing full-table scans more often.

- There is a lot of time spent waiting for db file scattered read, so I increase the buffer cache, and I don’t see the symptom any more. Because the query that was responsible for this wait time, previously did a full-table scan from disk, and now the table is smaller than _small_table_threshold, so the table is cached, and the query is now doing the same inefficient full-table scan but from memory.

- The latch free event is at the top, I increase spin_count (now a hidden parameter), the symptom disappears, but instead I have increased the CPU consumption and the length of the CPU’s run-queue.

Often the action that is done, after drawing conclusions from hit ratios or wait events, is to increase some initialisation parameter (that affects the whole instance and every session), which will often put more load on the lower layers (the operating system and hardware). When the symptoms appear at those layers, as high CPU consumption, long run-queue, paging, swapping and long disk-queues, and then the company buys more and faster hardware.

When the table (in the example above) grows (due to more rows or fragmentation) and becomes bigger than the threshold, then the algorithm for reading a large table is used, which is not to cache the table at all, and now the table needs to be read from disk every time (the problem is back).

Some companies buy faster disk or more memory when most of the data is already in the buffer cache, or they read most of the data from disk, but they buy a faster CPU (which may create a longer queue to the disks, so the whole query is now running more slowly than before).

Is the problem a resource problem or a request problem? Do I see the symptom or the problem?

Move focus from the queue and wait time to the counter and service time

The third method to optimise a database system is to look at the requests that arrive, what kind of service they ask for and the execution plan that they choose. It is the requests that are keeping the system busy. The system is the activities' environment. If the root cause has to do with the activity, then that is the place where a solution is needed. If the root cause is that the environment is too small for the optimised activities, then that is the place where a solution is needed.

In the previous method we focused only on wait time (even if a wait time in Oracle is more of a response time); here we will look at what it is that the users complain about – response time (users don't complain about low buffer cache hit ratio or shared pool latch contention). Wait time only exists when a queue is created, and the queue is only created when the resource that the request is demanding service from is busy. In this method we move our focus to what is going on at the counter. What kinds of requests are coming in? What is the resource spending its time on, and how does the execution plan look for the request?

Queueing theory in practice

This text is built on my experience and the inspiration I had from reading a chapter in Cary Millsap's forthcoming book, Optimizing Oracle Performance. If the rest of the book is as good as this chapter, this will be one of my most valuable books. I have simplified some of the terms and some are not mentioned at all, so any errors and deprecation is my and only my fault.

Queueing theory is an optimistic model about the behaviour in a queueing system. It is optimistic in the way that if the model says that it cannot be done, it can't; if it says that it is possible, then it may be achievable in reality.

The queueing system has one or more service providers (or service channels). A service provider can be a CPU, bank cashier (where people line up in one common queue), a one-lane bridge or an elevator. The service provider has the symbol m. You can have more than one queueing system for the requesters, like RAC (two or more instances of the same database), supermarket cashiers (where all of the cashiers have their own separate queue), or multiple bank offices (where every bank office has one single queue. The number of queueing systems has the symbol q.

In a queueing system we have requesters (who need service) and service providers (who provide service). A is the number of requests that arrive at the system. C is the number of completed requests. T is the time period. From this we get $\lambda = A/T$ (lambda), the system's arrival rate (how many requests arrive per time unit), and $\tau = T/A$ (tau), the mean interarrival time, and $X = C/T$, the system's complete rate or throughput.

The time it takes for a service provider to serve a request is called service time (S) and the service rate, μ (mu), is the number of requests that one service provider can complete per time unit. The time the request had to wait until the service provider was available is called wait time (W). For the same type of request the service time is always the same, but wait time can differ from 0 to X. The total time that it took from asking for service until getting it is called response time (R). So the formula for response time is $R = S + W$. Here I look only at one single request; you will find the real formulas in Cary Millsap's book. One big difference has to do with the expected wait time. We calculate the arrival rate and the service rate, so in a very simplified way, we can say that a system is starting to build a queue when the arrival rate exceeds the service rate. But that is not correct, the queue is built much earlier than that; because the arriving requests will not drop in with the exactly same interval (when I was a DBA/developer for the Stockholm Stock Exchange, we had days with 32000 transactions, on average 90 per minute, but it didn't mean that we had 90 transactions every single minute, it ranged from 0 to X).

A queue is created when a requester wants service but the service provider is busy serving an earlier request. The more requests in the queue, the longer the wait time; but the service time will remain the same. Remember that I'm talking about average response, service and wait time. If I am a bank manager and I have collected statistics for many years about how many customers are entering my bank, and how long they wait in the queue and how long they spend being served, then I can predict and be quite sure what my average customer will experience tomorrow. But I cannot predict with the same confidence how long time a request will take for one specific customer.

BAD * MORE = WORSE. If you have something that doesn't perform optimally, and you start to do more of it, you reach a point where it suddenly becomes worse. By optimising the users' behaviour, the application, the SQL, the network, the Oracle instance, the O/S and the hardware, we move this point further away, so we can handle more load and throughput until we reach it.

From a user, system owner and performance point of view, these two terms are also of interest: r_{max} (the maximum response time that the users can accept) and *p* (percentages of times when response time is below r_{max}).

References:

Optimizing Oracle Performance, Cary Millsap, Hotsos (O'Reilly books, not yet published, probably summer 2003)

Layers between the users and the data

```
USERS >----+  +--+                                        +--<        S Q L
           |  |  |                                        |           R   E
APPL       +--+  +----+       W A I T I N G      +--+  +--+           E   S
                      |                          |  |  |              Q   N
ORACLE   I D L I N G  +--+  +--+        +--+  +--+  +--+              U   O
                         |  |  |        |  |  |                       E   P
O/S                      +--+  +--+  +--+  +--+                       S   S
                                  |  |                                T   E
HW                                +--+                                +---R
```

A resource (layer) can be waiting for a request (the resource is idle), serving a request (which will be counted as service time) or waiting for a request from the layer below to complete (counted as wait time). Oracle records this wait and idle time (even CPU time spent in system/kernel mode is included here, so a wait time in Oracle is not a real wait time, it is a response time) in v$session_wait, v$session_event and v$system_event (time_waited), and the service time (CPU time spent in user mode) in v$sesstat and v$sysstat (CPU used by this session).

We can tune our systems by looking at one layer and try to optimise it. We often see the bottlenecks at the O/S, HW and database layers, but that doesn't mean the layer is the problem. We have one server, on that server we run 1-4 databases, every database has maybe 1-20 applications, and every application has X users. Every layer puts load on the layer below, so it is not strange that we see the symptoms on the lowest layers. These layers are also the most shared resources. It is much easier to focus and spend money on the lower layers because they are fewer (we have only 4 CPUs but 400 SQL statements plus 40000 lines of code that are using the CPU), the price is often fixed (to buy two more CPUs cost X dollars but what will it cost if our developers go through and clean up their and others' code) and we are hoping for a one-action-solves-all formula. But when we add more and faster resources to the lower layer, they are used by all incoming requests (good as bad). This is what I would call horizontal tuning, which should only be used in two situations: 1) Put out the fire (we know this is a fast fix but we must do it before we search for the root cause). 2) The root cause is that we have too few resources for this layer, so this is a permanent solution.

Vertical optimisation (the word optimisation is on a higher level than tuning, that's why I don't call it vertical tuning) looks at what goes through all layers? The SQL statements! They cut through all layers, put load and request service from them. Again, a database system without any SQL would be infinitely fast.

By catching these wait events (either how the V$ views looked before the query started and then after, or by using event 10046 with level 8) together with the CPU time, we can see where the majority of the response time is spent. If the response time (called *elapsed time* in tkprof) is 100 seconds, CPU time is 95 seconds and the remaining 5 seconds is spent on physical read, then we should concentrate on service time (if we focus on wait time, we can only reduce the query's response time by a maximum of 5 seconds). If the session is idle most of the time, then we know that we must start focusing on the application and the higher layers.

When the session is…	Which wait events
Idle (waiting for requests to arrive) SQL*Net message from client is what a user session normally waits for.	pmon timer (only for the pmon process) smon timer (only for the smon process) rdbms ipc message (for all the other background processes) dispatcher timer (for MTS dispatchers) virtual circuit status (for MTS shared servers) pipe get SQL*Net message from client SQL*Net message from dblink
Waiting for an activity inside Oracle Most of these wait events mean that the session is waiting for another session to complete its activity (LGWR to finish a log switch, or another session to commit a transaction). PL/SQL lock timer, is often placed by the idle events, but in my eyes, the session is not idle, because it cannot receive new requests. The session is waiting for an internal timer to expire, and until that happens, it will receive and serve no requests.	PL/SQL lock timer buffer busy waits checkpoint completed enqueue free buffer waits latch free library cache load lock library cache lock library cache pin log buffer space log file switch (archiving needed) log file switch (checkpoint incomplete) log file switch completion log file sync pipe put sort segment request undo segment extension
Waiting for response from a lower layer All these events are about file management (open, read and write).	control file sequential read db file parallel read db file scattered read db file sequential read direct path read direct path read (lob) direct path write direct path write (lob) file open

This table is not complete and it focuses mainly on the Oracle server (shadow/foreground) process.

References:

Tuning by layers, Steve Adams, IxOra

You have become a bank director

Let me make an analogy. You are the happy director of a bank (where your customers line up in one common queue). On average 10 (A) customers come every hour (T) to your bank. You have one cashier (m) and it takes him on average 2 minutes (S) to serve one customer. You have the goal that 80% (p) of the customers shall have completed their business within 5 minutes (r_{max}).

You hear that they will build a lot of apartments close to the bank. More customers = more business, if you can still give the same good service. You estimate that you can get twice as many customers. Can you still reach your goal?

You put the numbers in your Queueing theory calculation sheet and find that if you don't do anything then only 57% of your customers will have completed their business within 5 minutes, or 80% of your customers will have completed their business within 10 minutes. What alternatives do you have if you don't want to (cannot) accept this? Because accept it is one thing that you can do.

- Increase number of queueing systems (*q*), which means to build another bank and hope that half of the customers will choose that one. Then you will reach your goal in 81% (*p*) of the time.

- Increase the number of service providers (*m*), which means hire a second cashier and spread out that cost on your services, and you will reach your goal in 89% of the time.

- Decrease the arrival rate (λ), which means that the customers don't come so close after each other. This can be done in two ways:

    1. Decrease number of arrivals (*A*). Maybe you can get an ATM so less customers line up in the bank.

    2. Increase the time period (T). Maybe you can extend the opening hours and put that additional cost on your service.

All of these alternatives are trying to reduce the wait time, but that doesn't affect the service time. Remember that wait time is a function of service time and mean interarrival time. So what happens if we try to focus on what is going on at the counter and the requests that the customers have, instead of focusing on the customers who are waiting (we move from the symptom/effect to the cause).

- Decrease service time (the same as increase service rate). If the average service time can be decreased from 2 minutes to 1.5, then you can keep your old goal of 80% within 5 minutes. This can be done in three ways:

1. Working faster (faster CPU), for you it means training the cashier to do exactly the same work but faster.

2. Working less (shorter code path, this is something developers should do much more often, go through their old code and remove parts that are not needed any longer), can the customers be better prepared when they reach the counter and have everything ready? Teach the customers to only do business inside the bank that requires a cashier

3. Working smarter (optimise the code, can the same result be achieved in a better, cheaper and faster way), introduce a better computer system, have automatic money counting machine, tune internal processes and routines.

If you can decrease the average service time from 2 minutes to 1.5 minutes (a reduction of 25%) then you can still keep your old goal (80% of your customers shall have been served within 5 minutes).

Focus on response time and the execution plan with event 10046, level 8

A lot of papers have been written about the best trace event in Oracle: 10046. When a user is executing ALTER SESSION SET SQL_TRACE = TRUE; internally he sets this event with level 1 (and 0 to turn it off). This event can also be used with level 4 (to show the values of bind variables) and level 8 (to show the wait events). With level 12 the raw trace file contains both bind variable information and wait events statistics.

To run a 10046 trace set to level 8 and with timed_statistics set to TRUE, is one of the best things you can do when you are experiencing performance problems. You get detailed information about the response time and the execution plan.

The raw trace file:

```
PARSING IN CURSOR #1 len=48 dep=0 uid=26 oct=3 lid=26 tim=22149871117 …
select count(distinct c) from parent where pk=18
END OF STMT
PARSE #1:c=921875,e=975658,p=0,cr=71,cu=0,mis=1,r=0,dep=0,og=4,tim=…
EXEC #1:c=0,e=136,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=22149871443
WAIT #1: nam='SQL*Net message to client' ela= 7 p1=1111838976 p2=1 p3=0
WAIT #1: nam='db file sequential read' ela= 9008 p1=6 p2=3364 p3=1
WAIT #1: nam='db file sequential read' ela= 230 p1=6 p2=3365 p3=1
WAIT #1: nam='db file sequential read' ela= 13002 p1=6 p2=3080 p3=1
FETCH #1:c=15625,e=81941,p=3,cr=3,cu=0,mis=0,r=1,dep=0,og=4,tim=22149953504
WAIT #1: nam='SQL*Net message from client' ela= 603 p1=1111838976 p2=1 p3=0
FETCH #1:c=0,e=5,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=22149954477
WAIT #1: nam='SQL*Net message to client' ela= 2 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 3780574 p1=1111838976 p2= …
STAT #1 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT GROUP BY (cr=3 r=3 w=0 …
STAT #1 id=2 cnt=1 pid=1 pos=1 obj=7330 op='TABLE ACCESS BY INDEX ROWID …
STAT #1 id=3 cnt=1 pid=2 pos=1 obj=7332 op='INDEX UNIQUE SCAN PK_PARENT …
```

Formatted with tkprof in 9.2:

```
select count(distinct c) from parent where pk=18

call    count     cpu  elapsed       disk      query    current     rows
------- -----  ------ -------- ---------- ---------- ----------  -------
Parse       1    0.92     0.94          0          0          0        0
Execute     1    0.00     0.00          0          0          0        0
Fetch       2    0.01     0.08          3          3          0        1
------- -----  ------ -------- ---------- ---------- ----------  -------
total       4    0.93     1.02          3          3          0        1

< Response time = elapsed, service time = cpu, wait time = elapsed – cpu >

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 26  (SLASK)

Rows  Row Source Operation
----  ---------------------------------------------------
   1  SORT GROUP BY (cr=3 r=3 w=0 time=81908 us)
   1   TABLE ACCESS BY INDEX ROWID PARENT (cr=3 r=3 w=0 time=71081 us)
   1    INDEX UNIQUE SCAN PK_PARENT (cr=2 r=2 w=0 time=57916 us)(id 7332)

Rows  Execution Plan
----  ---------------------------------------------------
   0  SELECT STATEMENT   GOAL: CHOOSE
   1   SORT (GROUP BY)
   1    TABLE ACCESS   GOAL: ANALYZED (BY INDEX ROWID) OF 'PARENT'
   1     INDEX   GOAL: ANALYZED (UNIQUE SCAN) OF 'PK_PARENT' (UNIQUE)

Elapsed times include waiting on following events:
  Event waited on                             Times   Max. Wait  Total Waited
  ----------------------------------------   Waited  ----------  ------------
  SQL*Net message to client                       2        0.00          0.00
  db file sequential read                         3        0.01          0.02
  SQL*Net message from client                     2        3.78          3.78[1]
< [1]this wait event doesn't belong to this cursor – tkprof reports wrong >
```

In 9i, tkprof will summarise the wait event statistics for every cursor below the explain plan. Even if tkprof summarises the raw trace file, it is still a big advantage to be able to read, interpret and analyse the raw trace file directly.

Start with tuning what is important for the business. A customer had a batch job that took seven hours to complete, and after that they needed to take down the database for a cold backup. But because the batch job took so long, the backup didn't finish until the employees had already arrived at work. They were frustrated that the database was still unavailable. I changed the code in the batch job slightly and added an index, and the batch job completed in four minutes. The customer said, "We don't need it to be so fast. No other queries need that index, and without the index the batch job finished in two hours. So we don't want that index."

References for further readings

MetaLink note: 171647.1: Tracing Oracle Applications using Event 10046
MetaLink note: 21154.1: EVENT: 10046 "enable SQL statement tracing (including binds/waits)"
MetaLink note: 39817.1: Interpreting Raw SQL_TRACE and DBMS_SUPPORT.START_TRACE output
Oracle System Performance Analysis Using Oracle Event 10046, Cary V Millsap & Jeffrey L Holt (http://www.hotsos.com/dnloads/1/10046a/index.html)

Good views in 9i that can be used:

- V$SQL (avoid V$SQLAREA)
  - rows_processed > *fetches* >= executions > parse_calls

    In a healthy system, we parse once and execute many, and we use array fetching to decrease roundtrips.
  - disk_reads, buffer_gets, rows_processed, executions

    Excellent to find suspicious statements (low buffer cache hit ratio, performs many buffer_gets per rows_processed, high disk_reads per execution).
  - users_executing. To find all statements that are executing (more correctly, "pinned") just now, execute SELECT * FROM v$sql WHERE users_executing > 0.
  - *cpu_time, elapsed_time* (in microseconds if timed_statistics = TRUE or statistics_level = BASIC).
  - *child_latch* (= v$latch_children.child#, where name='library cache', new in 9.2)
- V$SQL_PLAN_STATISTICS_ALL shows the execution plans that were used for the cursor, with excellent information for every row source operation/step: reads, writes and elapsed time, access and filter predicates, output rows and SQL working memory management statistics. Requires that statistics_levels is set to ALL (or that _rowsource_execution_statistics = TRUE).
- V$SEGMENT_STATISTICS, V$SEGSTAT & V$SEGSTAT_NAME, shows a lot of valuable information per segment: logical reads, buffer busy waits, db block changes, physical reads, physical writes, physical reads direct, physical writes direct, global cache cr blocks served, global cache current blocks served, ITL waits, row lock waits. Requires statistics_level = TYPICAL or _object_statistics = TRUE.
- V$ENQUEUE_STAT (X$KSQST)
  - CUM_WAIT_TIME, shows for how long (in microseconds) sessions have been waiting for this type of lock.

- V$LATCH, V$LATCH_PARENT, V$LATCH_CHILDREN
    - WAIT_TIME, shows in microseconds how long sessions have been waiting for this latch.
- V$SYSSTAT, V$SESSTAT
    - CPU used by this session
- V$SESSION
    - status, last_call_et, sql_hash_value, row_wait_%
- V$SESSION_WAIT, the best view to see what's going on just now in the instance. This view should be one of the first views to look at; when someone asks, "what is going on just now" or "what is my session doing now". You need to understand how this view is updated and when it is updated.
- V$LOCK, the fastest way to find out if anyone is waiting for a lock and who the blocker is, is to execute SELECT * FROM v$lock WHERE request > 0 OR block > 0 ORDER BY type, id1, id2, ctime DESC;

So what do I want you to do:

- Optimise the code while you are writing it (it is easiest and cheapest to change).
- Don't focus on instance hit ratios and use it as a foundation for your tuning.
- Look at session buffer cache hit ratios (v$sess_io) to find out if all sessions are reading a lot from disk or only a few.
- Look at statement buffer cache hit ratio (v$sql) to find the statements that are responsible for many disk reads.
- Be suspicious of statements that do a lot of buffer gets per execution or buffer gets per processed row.
- Don't increase the buffer cache if you don't know why you are doing it, or just guessing.
- Be sure that you don't introduce new problems by increasing the buffer cache:
    - log file switch (checkpoint incomplete)
    - DBWR has problems keeping up
    - Latch contention
    - Taking too much memory from the operating system
    - Hiding inefficient SQL that before did disk reads, and are now doing inefficient logical reads
- Use the multiple buffer pool feature to divide tables into the KEEP (here we want to see high buffer cache hit ratio), RECYCLE (here we want to see low buffer cache hit ratio) and DEFAULT pool
- Use event 10046 with level 8 to find expensive SQL, and focus on response time and execution plan.
- Start focusing on queries, transactions, jobs and applications that are important for the business, are executed frequently and take a long time to complete.
- Check if you can decrease the amount of work. If the job can be run less frequently or not at all, then you save a lot of resources, which will speed up other requests.

About the Author

Joakim Treugut is Synergy's chief Oracle investigator and problem solver, and a member of the Oaktable Network. He has inside knowledge of how Oracle works behind the scenes - thanks to 18 years of development and database experience. He started to work as a developer and DBA for the Stockholm Stock Exchange where he stayed for 13 years. After that he worked for Oracle Support in Sweden for 5 years. During his first year he was selected as an "Architect of Success". He soon became the specialist on performance problems and Oracle internals. He was the team leader of the Server kernel team, responsible for database performance and ORA-600/7445 errors. He was also a member of the European SQL tuning team.