

Oracle Workload Management Using Time Based Optimization Techniques

Author: Andy Rivenes

Copyright © 2003, AppsDBA Consulting, All Rights Reserved.

Introduction

For the Oracle performance analyst, “time based optimization” is the holy grail of performance optimization. Time based optimization is the technique of analyzing a particular interval of time, and identifying the components that consume resources during that time. This technique works equally well for capacity planning at the system level or response time optimization at the session level.

In computer based systems, system capacity is generally categorized into four areas: CPU, memory, disk, and network. In an Oracle database, workload is typically motivated by the execution of SQL, but increasingly can also be created by Java and PL/SQL programs that may or may not invoke SQL calls to the database. In all these cases CPU is required to perform the work, with memory, disk, and network components supporting the production of this work. Therefore, workload can be considered as the measurement of the “rates” of work being performed. In an Oracle database we can think of CPU time in the context of “service time”, since it is CPU that drives our workload, with the other components of system capacity as additional rates to be measured that contribute to the ability to provide service time. The Oracle database has been instrumented with events that account for all¹ time not spent in “service time”. These “wait” events are due to the costs of synchronization of access to common resources². Combined, service time and wait time can provide a complete picture of the time consumed in an Oracle database.

At the system level, time based optimization can be used to quantify database server workload and predict system capacity. Gunther³ defines utilization to be the time the system was busy during a measurement period. In an Oracle database service time can be measured for a specific time interval with the v\$sysstat statistic “CPU used by this session”. Gunther⁴ further defines throughput as a direct measure of the number of completions. In an Oracle database this can be quantified as the number of “logical I/Os”⁵ performed for a given time interval. Capacity then, is the maximum amount of “service” that a

¹ It appears that this is actually a misleading statement. Apparently not all events are instrumented hence the NULL event, but for the purposes of this discussion I ask the reader to accept that actual service time is well instrumented and that all other time is wait time of some sort.

² See the YAPP paper discussion of synchronization for a full description.

³ Practical Performance Analyst, Neil Gunther, Pg. 42

⁴ Practical Performance Analyst, Neil Gunther, Pg. 42

⁵ I will concede that the cost of all LIOs are not the same, but the LIO is a good measure of database throughput when used in conjunction with CPU usage. In an Oracle database data is retrieved through LIO calls, and most other work performed by the database is motivated by LIO calls.

system can perform, and workload is the amount of capacity that a database consumes. Therefore, since we can quantify the amount of service time our system consumes, and the throughput based on that service time, we can begin to make capacity predictions as we observe these quantities over time.

Based on our understanding of service time and wait time, at the session level we are measuring the response time of one or more transactions. Time based optimization can be used to quantify this response time because we can use event 10046 level 8 or higher tracing data to identify all of the time spent in a database server session. The results based on this information are astounding, and the time savings produced by identifying the largest time components of a transaction and then focusing on them, provides the greatest return on investment for the performance analyst. Another benefit of this technique is the ability to show the savings achieved during the optimization process because the response time can be quantified before and after.

This paper will explore each of these approaches to time based optimization, and will hopefully convince the reader to begin using these techniques to help solve their Oracle performance and capacity problems.

Background

Oracle DBA practices have historically been rife with misinformation. Several common DBA myths have included maintaining high buffer cache hit ratios, keeping segments in very few extents, and periodically rebuilding the database. There are certainly many others as well. Many times performing these tasks did improve performance, but not because the tasks cured some ill. The “conventional wisdom” of the industry has evolved to include tasks that have cured some problem at some point in time, but have no basis in fact. It had worked, so it must be the right thing to do. In fact, it has been shown that none of these kinds of “tasks” normally need to be done, or at least not for the reasons being given. The problems solved by these tasks should have been solved in other ways, it just happened that in some cases a side effect of these tasks was immediate relief from a problem. Notice we said immediate because most “conventional wisdom” doesn’t address the underlying problem.

“Conventional wisdom is opinion that has been repeated enough over time to become accepted as fact. While sometimes it can be fact, most times it is not.”⁶ Oracle performance tuning has also experienced the same conventional wisdom problems, and has gone through several “phases” over the years. Ultimately the goal has always been identifying some “thing” that is a bottleneck and trying to optimize it. Unfortunately, past methodologies have not been precise enough to allow the tuner to easily find the problem.

⁶ The Picture Perfect Pitcher, Tom House and Paul Reddick

Initially there was focus on “ratio-based” tuning methodologies. Ratios like the database buffer cache hit ratio were used to determine whether the database or a particular process was “tuned”. Then came “wait-based” tuning. This focused on determining what a process was waiting on and then trying to reduce it. With the publication of the YAPP paper, the idea of focusing on response time was introduced. The key concept added by the YAPP paper was the introduction of service time into the equation. Now we could think of response time in terms of service time as well as wait time. The strength of the response time model is that it can be applied at the system level as well as the transactional level. At the system level we can view response time, or throughput, in relation to computing capacity, and at the transaction level we can view response time in terms of the end-user experience.

Ratio Analysis

Ratio based analysis fails because it invariably fails to answer the question being asked, and it tends to provide unreliable information⁷. A classic example of this is the database block buffer cache hit ratio (BCHR). The ratio is typically calculated as $(1 - \text{physical reads} / \text{logical reads}) * 100$. The problem with this ratio is that it fails to provide reliable information. If you ask most DBAs why the BCHR is important they will probably answer that it is a measure of how well database blocks are being cached and that the higher the ratio the better the database is performing. In fact, the ratio is simply a measure of the proportion of logical reads that did not require a physical read. Presumably, the higher the number the fewer physical reads that are taking place. However, the real motivation for looking at a buffer cache hit ratio should not be to measure the amount of physical I/O being performed, but to measure the amount of repeated physical I/O being performed for a given database block. The problem with the BCHR ratio is that it lumps all blocks together. It fails to answer the question.

Let’s review Oracle basics here. In order to access a database block it must be read into memory first. This ultimately requires a physical I/O. If you only access the block once then the BCHR is not going to be very good since you will have one logical read that motivates one physical read. However, if you access a block many times, and if it stays in memory and does not have to be re-read from disk, then it’s BCHR will be much better. The problem is that to measure this on a system wide basis provides very little useful information because it combines the necessary physical I/O (i.e. to retrieve the block in the first place), with possibly unnecessary physical I/O (i.e. repeated physical I/O because the block was flushed out of the buffer cache). In addition, the ratio can be easily skewed to look favorable by excessive LIOs against blocks in memory (e.g., [Millsap 2001]).

If we could measure and track logical and physical reads on a block basis then a BCHR would provide a very different and interesting picture. On a block level basis, we could identify the repeated physical block I/O. In many cases, the 80-20 rule applies, 20% of the database blocks cause 80% of the I/O. If these highly volatile blocks are kept in memory and not flushed by other less volatile blocks

⁷ Millsap, 2001- OAUG DB SIG

then the “repeated” physical I/O would be reduced. In fact, this was the motivation behind Oracle’s multiple buffer pools feature. From a workload perspective this would reduce the file I/O rates and from a response time perspective the physical I/O time could be reduced if fewer physical I/Os take place.

So, ratios can prove to be “indicators” of something. The trick is to properly define the something and then to recognize that the indicator may or may not be providing reliable information. There are numerous additional examples available to further reinforce that ratios are unsuitable as a basis for performance tuning Oracle databases.

Wait Events

Many Oracle DBAs thought that wait events were their performance tuning “holy grail”. In fact, this may be the most popular performance tuning process currently in practice. The problem with wait event tuning though, is that it lacks context. As we have already stated, response time equals service time plus wait time. Service time equates to CPU time, and wait time equates to time spent waiting for some event to occur. Since an individual transaction’s response time is made up of both service time and wait time, focusing strictly on wait time misses what can be a large part of the response time component.

To illustrate this we can review the “Rates and Waits” section from an event 10046 level 8 trace file that was summarized by an event 10046 trace parser⁸. If we take a closer look at the components that make up this transaction’s response time during the observed interval we see the following:

<u>Event</u>	<u>Percentage</u>
SQL*Net message from client	68.89
CPU Service	24.42
unaccounted-for	6.22
SQL*Net more data to client	0.46
SQL*Net message to client	0.00

In this particular case, if we did not consider CPU Service, which we wouldn’t in a strictly wait event analysis, we would miss over 24% of the total response time for this transaction.

This problem becomes even more pronounced at the system level. Wait time at the system level is not directly measurable. As has been stated, for any given session, response time will be composed of service time plus wait time. At the system level, it’s not response time that we are measuring, but throughput or the capacity to perform work. In other words, the measurement interval is simply elapsed time, and the utilization or workload will equal service time divided by elapsed time.

⁸ The full summary of this example trace is presented in the Session Level Time Based Optimization section.

As we have stated, wait time cannot be derived directly. The problem is in measurement. Oracle will record total wait times, through the `v$system_event` view, but only for events that sessions waited on. There is no concept of “system” wait time in Oracle’s measurements. Wait time in this context is infinite. This occurs because there is no limit to the number of sessions that can connect and disconnect during any interval period. However, each session that is connected during the interval will contribute to the total wait time of the various wait events.

The effect of this is that while system level elapsed time is still composed of service time plus wait time, only CPU time and elapsed time can be derived directly. Wait time measured by Oracle at the system level, because it is composed of all sessions, will be the sum of all session elapsed times minus CPU time. At the system level this number is of no use. Another way to look at this is that wait time at the system level is simply time spent not doing “work” (e.g. time not spent in service time or doing useful work). This time may be due to synchronization time within the database or other internal (e.g. disk service time) or external (e.g. network or user think time) database server time.

System Level Time Based Optimization

System level time based optimization can be measured through the use of the system level statistic “CPU used by this session”. This statistic shows the cumulative CPU used by all database sessions since instance start. At the system level this statistic is obtained from the view `v$sysstat`. Based on this information we can quantify the CPU usage of an Oracle database, and therefore identify its capacity requirements.

This statistic is reported in 100ths of a second and may be on the low side because time slices of less than 1/100th of a second will not be recorded. It may also be high or low because Oracle only updates this statistic when a user call completes. This means that long running queries and PL/SQL processes, which use CPU, will NOT show up in the CPU statistic until the query or outermost PL/SQL block completes. This can therefore skew a given interval period if one or more calls complete that spanned one or more previous periods. The previous periods will also have lower times than they should. This can be detected to some extent by the statistic ‘CPU used when call started’. Oracle will mark the CPU time used when a call starts in this statistic. Unfortunately this statistic is only of value at the session level, since the system level statistic is updated every time another session starts a call.

Another issue to consider is that the `init.ora` parameter “`resource_limit`”, when set, has the side effect of causing “CPU based statistics to be updated more frequently than usual in order to ensure CPU resource limits are not exceeded.”⁹

⁹ Oracle Corporation, Note: 30800.1, “Reference Note for Init.Ora Parameter “`RESOURCE_LIMIT`”

Database CPU utilization can also be correlated to the operating system. This can provide additional information on CPU usage and may pinpoint otherwise unnoticed CPU usage. On UNIX, sar can be used to gather operating system level information. The following report excerpt was gathered with SYSMON¹⁰ on a 4 CPU HP-UX machine running a single Oracle 8.1.7.2 database. The collection interval was set to one hour, the sar utility was run concurrently, and this particular interval spanned a 24 hour collection period:

```

SYSMON Rates and Waits Database Report

Interval: 31-JAN-2003 00:46 - 23:47

ORACLE Database Version
-----
Oracle8i Enterprise Edition Release 8.1.7.2.0 - Production
*****
Service Time Section
*****

Instance CPU Utilization

Interval   Total CPU
Elapsed   Elapsed
Time(Sec) Time(Sec)   DB CPU   DB CPU
Time Used(Sec) Avg Use(%)
-----
      82,870      331,480          69,944      21.10

Instance CPU Usage

Parse CPU   Parse   Rec. CPU   Rec.   Other CPU   Other
Time(Sec)  CPU %   Time(Sec) CPU %   Time(Sec)  CPU %
-----
   5,592.08   8.00   20,559.04  29.39   43,792.96  62.61

OS CPU Utilization

OS CPU   OS CPU Average Maximum   OS Usr  OS Usr  OS Usr
Avg. Util% Max Util% OS Load OS Load  Avg(%)  Min(%)  Max(%)
-----
    23.54    66.00    0.25    1.00    21.38    3.00    61.00

```

The key items to note are the “DB CPU Avg Use(%)” number and the “OS Usr CPU Avg(%)” number. The first number is the average service time utilization that Oracle measured for the given time interval. The second number is the average user mode CPU utilization that was recorded through “sar” at the UNIX level during the same time interval. In general, these numbers should be the same, in practice they are usually very close for a dedicated database server. When they are not, it usually indicates some other OS level resource consumption, or some database measurement error.

¹⁰ SYSMON is a workload based interval monitoring tool.

Over time this information can be displayed in the following manner:

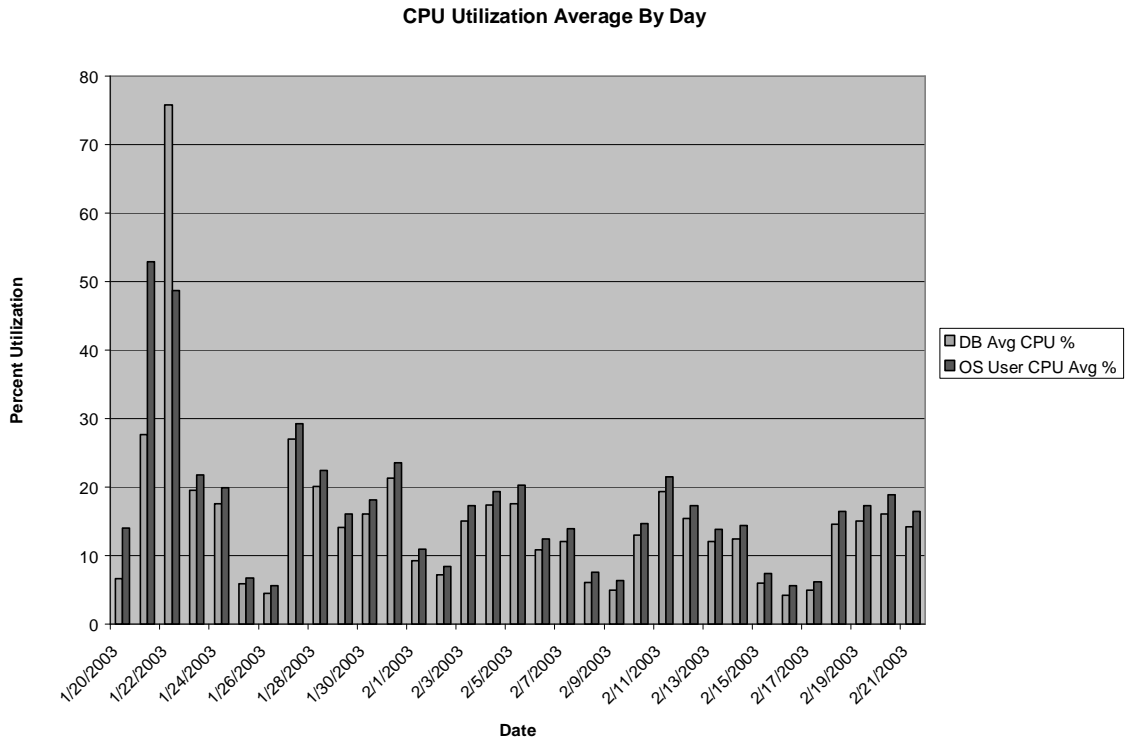


Figure 1. Average daily database and OS User CPU utilization (hourly collection interval)

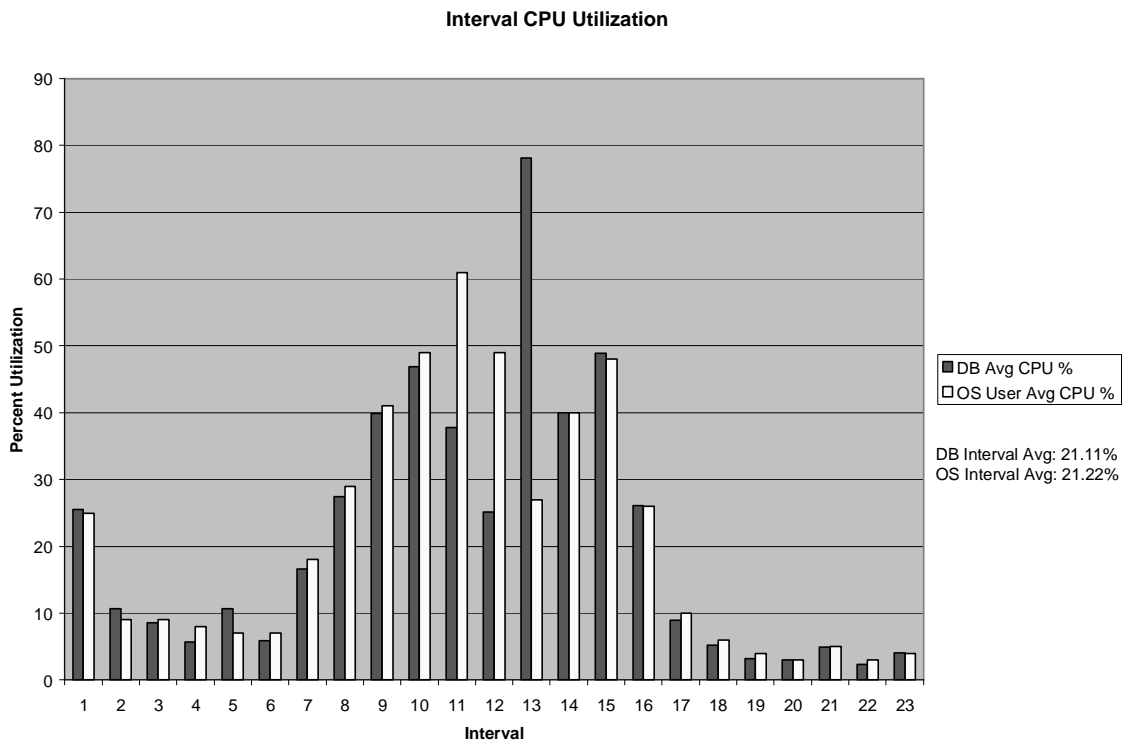


Figure 2. Average hourly database and OS CPU utilization (hourly collection interval)

Measurement Skew

As you can see from the graphics, measurements can skew across one or more measurement intervals. We have already described the issues involved with sampling from the Oracle side. As illustrated, this shows up across intervals as discrepancies that generally even out, for the most part, over time.

Session Level Time Based Optimization

Time based optimization can be applied to Oracle transactions or “sessions” through the use of event 10046 tracing. Unfortunately, the tools available to interpret the data have been the biggest roadblocks to its wider adoption. Initially the only Oracle supplied tool to interpret event 10046 traces was tkprof¹¹. The problem with tkprof is that it is really meant to allow the displaying of individual SQL statement executions. It is not until the 9i versions of tkprof that any meaningful information is available about wait times, and what is available is not sufficient, or many times even accurate¹², in helping to interpret event 10046 level 8 and above traces. Oracle has very recently released a new event 10046 trace interpretation tool that shows new promise in helping to interpret event 10046 traces. The tool is called TRCANLZR (Trace Analyzer), and it will parse an event 10046 trace file into a database schema and create a summary report.

There are currently only two other companies that I know of supplying tools to interpret event 10046 traces. They are Hotsos, LLC and the Hotsos Profiler, and Ubttools and the itrprof SQL Analyzer. In addition, there is not much documentation available, from Oracle or anyone else, explaining how to interpret the event 10046 trace file. There are Oracle documents explaining some of the key entries in the file, but not all of them, and there is no information about how the different entries inter-relate within the trace file. However, this should not discourage the performance analyst from exploring the information that is available, and as more analysts recognize the overwhelming benefit of event 10046 tracing more information and analysis tools will surely become available.

So, how does an event 10046 trace file enable us to use time-based optimization techniques? The key is in the ability to identify an interval and to be able to summarize this information into service time and wait time components, or “Rates and Waits”.

¹¹ There is an Oracle internal tool called TRCSUMMARY, but this is not available outside of Oracle.

¹² In version 9.0 (at least) tkprof fails to correctly report statement level waits for statements with multiple cursors. Only waits associated with the initial cursor are summed.

This is best illustrated by the following example:

SESSION INFORMATION

Session SID: = 20

Connection Summary

Session start: = 11-26-2002 15:29:40
Interval start (t0) = 756122956
Interval end (t1) = 756123390

Interval duration = 4.34s
SQL cursors found = 110
Distinct SQL statements = 64

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

Action	Count	CPU	Elapsed	PIO Blks	LIO Blks	Consistent	Current	Rows
PARSE	99	0.03	0.07	0	6	6	0	0
EXEC	484	0.15	0.14	0	16	16	0	0
FETCH	528	0.88	0.85	0	8024	7740	284	33093
Total	1111	1.06	1.06	0	8046	7762	284	33093

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

Action	Count	CPU	Elapsed	PIO Blks	LIO Blks	Consistent	Current	Rows
PARSE	9	0.00	0.00	0	0	0	0	0
EXEC	11	0.01	0.03	0	0	0	0	1
FETCH	9	0.00	0.00	0	34	34	0	9
Total	29	0.01	0.03	0	34	34	0	10

INTERVAL RATES AND WAITS (service time + wait time)

Event Name	Total Event Time (sec)	% Event Time	Total Events	Avg Event Time(sec)	Max Event Time(sec)
SQL*Net message from client	2.990000	68.89	1413	0.002116	0.880000
CPU Service	1.060000	24.42	1140	0.000000	0.000000
unaccounted-for	0.270000	6.22	1	0.000000	0.000000
SQL*Net more data to client	0.020000	0.46	696	0.000029	0.020000
SQL*Net message to client	0.000000	0.00	1413	0.000000	0.000000
Total	4.340000	100.00			

This information was derived from a session trace with event 10046 level 8 tracing enabled. For the observed time interval, this section outlines the time consumed and the components that make up this time. The "interval duration" in the session information section displays the total time used by this interval. The service time, which in Oracle is CPU time, and the wait time, all "wait events" summed by type, are displayed and totaled. Notice that this total is the same as the interval duration time. There is a special item called "unaccounted-for", which will be discussed more fully later, that accumulates any discrepancy between the observed interval duration and the sum of the rates and waits.

tkprof

The best that tkprof can offer is the following, and this requires 9i or higher. Previous versions will not show any wait information.

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

call	count	cpu	elapsed	disk	query	current	rows
Parse	99	0.03	0.07	0	0	0	0
Execute	484	0.15	0.14	0	8	0	0
Fetch	528	0.88	0.85	0	7740	284	33093
total	1111	1.06	1.06	0	7748	284	33093

Misses in library cache during parse: 3
Misses in library cache during execute: 2

Elapsed times include waiting on following events:

Event waited on	Times Waited	Max. Wait	Total Waited
SQL*Net message to client	1408	0.00	0.00
SQL*Net message from client	1408	0.15	2.97
SQL*Net more data to client	696	0.01	0.02

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call	count	cpu	elapsed	disk	query	current	rows
Parse	9	0.00	0.00	0	0	0	0
Execute	11	0.01	0.03	0	0	0	1
Fetch	9	0.00	0.00	0	34	0	9
total	29	0.01	0.03	0	34	0	10

Misses in library cache during parse: 0

Elapsed times include waiting on following events:

Event waited on	Times Waited	Max. Wait	Total Waited
SQL*Net message to client	5	0.00	0.00
SQL*Net message from client	5	0.01	0.02

101 user SQL statements in session.
9 internal SQL statements in session.
110 SQL statements in session.

Notice that since we have no interval duration information, we have no easy way to compare response time to service time and wait time. In addition, we have discovered that tkprof ignores subsequent wait event timing for the same SQL statements with multiple cursor versions. This leaves the information available from tkprof as inadequate to perform effective time based optimization.

trcanlyzr

The TRCANLYZR utility has recently been released by Oracle in support of Oracle Applications systems, but works equally well for any event 10046 trace. This utility retains much of the “tkprof” look, but does a much better job of summarizing event 10046 trace data and provides detailed cursor, SQL explain plan, and event information. Another big advantage of this tool is that it loads the trace file into a database schema, so reports can be run on the parsed trace file repeatedly and trace files can be kept historically for later comparisons. This tool is a big breakthrough in helping performance analysts analyze event 10046 information.

Interpreting a 10046 Trace File

There are several Oracle “Notes” available from Metalink that describe the major components of 10046 trace files. The most comprehensive is Note: 39817.1, “Interpreting Raw SQL_TRACE and DBMS_SUPPORT.START_TRACE output”. A second note, Note: 62160.1, “Tracing Sessions in Oracle7/8”, also provides some “usage” information about the trace file make up. Other than these notes, there is no other information that I know of, describing the architecture of the trace file. It is up to the user to determine how Oracle writes the file, the hierarchy and depth inter-relationships of the traced statements, and the other information written to the file. I believe that as more analysts recognize the value of this information, that more tools and information will become available, and that event 10046 tracing will become a standard approach to session performance analysis.

Category unaccounted-for

The category “unaccounted-for” is basically a catch all for any discrepancy between the observed interval time, and the sum of service time plus wait time for the given session within the trace file. Typically, there is a small amount of time due to “timing boundaries”¹³ within Oracle. This is usually most pronounced on Oracle systems prior to 9i where the timing granularity was limited to centiseconds. The second common cause of unaccounted-for time is system latency. This usually shows up as time spent on the CPU run queue, waiting to run. This category is mentioned in the Oracle Note: 62160.1 and on the

¹³ See the Oracle Reference guide description of the statistic “CPU used by this session” for an explanation of the issue of timing boundaries.

Hotsos.com web site. I know of no other references to this “category”, but this is most likely due to the lack of ability to even recognize this time, let alone investigate it.

I have also recently become aware of a mathematical method called quantization¹⁴ that may produce a fairly accurate estimate of the amount of timing error that is caused by this phenomenon. Currently, Cary Millsap of Hotsos, LLC is working on methods that may allow this technique to be used to identify the amount of measurement error that is occurring (see Hotsos 2003 Symposium notes for more information).

Conclusion

To conclude, time based optimization provides the Oracle database tuner with the best methodology currently available for solving Oracle performance problems. At the system level, interval based analysis of v\$sysstat CPU statistics provide service time information to allow the performance analyst to make capacity and throughput projections. At the session level the use of Oracle event 10046 level 8 or higher tracing provides the performance analyst with the best source of information necessary to categorize response time into service time and wait time. This enables the performance analyst to target the largest time components for optimization, with the goal being a reduction in response time. In addition, with a “Rates and Waits” based metric, a quantifiable measurement can be made, before and after, to show the actual improvements possible.

References

- MILLSAP, C. 2001. *Why a 99%+ Database Buffer Cache Hit Ratio is Not Ok*. Hotsos LLC.
- MILLSAP, C. 1999. *Performance Management: Myths & Facts*. Oracle Corporation.
- MILLSAP, C. 1998. *The System Architect’s Essential Role in Open Systems Implementations*. Oracle Corporation.
- GUNTHER, N. 2000. *The Practical Performance Analyst*. Choice Press, Lincoln, NE.
- KOLK, A.; YAMAGUCHI S.; VISCUSI, J. 1999. *Yet Another Performance Profiling Method (Or YAPP-Method)*. Oracle Corporation.

¹⁴ Millsap, Hotsos 2003 Symposium

HOUSE, T.; REDDICK, P. 2003. *The Picture Perfect Pitcher*. Coaches Choice, Monterey, CA.

Note: 224270.1, TRCANLZR.sql – Trace Analyzer – Interpreting Raw SQL Traces generated by EVENT 10046, Oracle Corporation, 07-FEB-2003

Note: 39817.1, Interpreting Raw SQL_TRACE and DBMS_SUPPORT.START_TRACE output, Oracle Corporation

Note: 62160.1, Tracing Sessions in Oracle7/8, Oracle Corporation

Oracle9i Database Reference, Release 2 (9.2), Statistic Descriptions – “CPU used by this session”, Oracle Corporation, Part No. A96536-01

Hotsos 2003 Symposium, Proceedings CD, Hotsos, LLC.

ORACLE7 I/O Monitoring and Tuning, Andy Rivenes and Neil Jensen, Lawrence Livermore National Laboratory