ABSTRACT

Early Oracle performance tuning practitioners used a large number of performance ratios to help optimize their databases. In the mid-1990’s session wait event analysis was born allowing direct Oracle contention identification. The next performance analysis frontier is response time analysis. Response time analysis focuses on quantitatively understanding performance pain.

Yes there is always a performance bottleneck. But sometimes we don’t care because performance is OK. What is missing from both Ratio Based and Wait Event Based analysis is quantifying user irritation and its components. Oracle response time analysis enables response time measurement, bottleneck validation, user irritation quantification, and improved tuning focus. Therefore, response time analysis is absolutely key to fully and efficiently optimize Oracle based systems.

There have been many challenges with measuring Oracle response time and its components. For example, Oracle does not mark a transaction like a transaction monitor, response time definitions are perspective based, lack of wait event instrumentation, and Oracle’s poor operating system resource instrumentation facilities. However, even with these challenges, response time analysis can be performed. In fact, response time and its components can be accurately calculated at the system and the session level…and using only data from Oracle’s virtual performance views.

This paper is all about understanding, using, and taking advantage of the next Oracle performance management frontier; response time analysis.

INTRODUCTION

Every system has a performance limiting bottleneck. That is, there is a specific reason why something will not run quicker. For Oracle based systems the bottleneck could be latching, user think time, or operating system CPU resources, just to name a few. If there were not a performance bottleneck, the system would run infinitely fast. This of course is impossible and therefore we can confidently state that there is always a performance bottleneck [13].

But our users may not care if there is a bottleneck because performance is acceptable. Sure the CPU may be the bottleneck, but if users are satisfied then they don’t care. If we sift through all the rhetoric, it comes down to irritation. Current performance is only an issue and therefore the bottleneck is only an issue when users are irritated.

This paper’s overall objective is to bring you to a place where you can validate the real bottleneck (not just Oracle wait events) [7], determine if users are irritated by the bottleneck, and where to focus your performance optimization efforts. But before we can do that, there are a few things we need to cover.

EVOLUTION OF ORACLE PERFORMANCE ANALYSIS

There are many approaches, methods, techniques, and tools to optimize Oracle systems. Because of history, familiarity, and economic incentives there is always a resistance to develop, influence, and use new superior approaches to optimize Oracle.

Besides SQL tuning, when Oracle systems where first tuned people simply added more memory to the SGA or sort area, perhaps moved database files around (if there where more then a few disks). But that was about it. And because Oracle systems were relatively small, this was acceptable.

However, as Oracle began to become accepted as a mainstream database product, Oracle based systems began to increase in size. Simply adding more memory or moving a few database files around was not good enough anymore.
As a result, the now classic approach to Oracle tuning was born. This classic approach is typically called *Ratio Based Analysis*. It is based upon the use of and the familiarity with a large number of performance ratios. For example, the number of sorts to disk divided by the number of sorts. Obviously one wants to minimize the sorts to disk, so a ratio near zero is optimal. Given enough performance ratios that cover the many areas of Oracle, given enough experience using the ratios, and given enough application specific experience, this approach will work very well.

There have been and still are many performance papers, presentations, and books regarding the use of performance ratios. These publications are typically very lengthy because of the number of ratios that must be covered. Because of this complexity, the people who have mastered this approach have secured a very comfortable living. Think about it. They can use ratio based analysis very effectively so they produce quality work, but because of the complexity involved, longer and more complex consulting engagements are needed. This results in an aura of "wow" surrounding these individuals. This also increases their consulting rates and increases publication sales, further increasing the "wow" factor. This is a comfortable situation for the chosen few who use and market this Oracle optimization approach.

But the situation has changed dramatically over the last few years. During the Oracle 7 years, Oracle began instrumenting the kernel code to include *triggers or sensors* that reported when an Oracle process was trying to get something (e.g., latch, physical i/o, cached block, enqueue, etc.) but that something was not immediately available. Anyone who could query from three "v$" views could tell exactly what Oracle processes where waiting for...with precision and with no ambiguity. These virtual performance views are the *session wait or wait event* views; *v$system_event*, *v$session_event*, and *v$session_wait*. This method of Oracle contention identification is known as *session wait event based performance analysis* [7].

The introduction of the session wait views provided a superior Oracle optimization approach compared to ratio based optimization. But because of it's simplicity, *anyone* could do it. Among other things, the lack of earning potential did not make this approach the favored approach with the ratio based community.

To summarize, Oracle's session wait views provides for a quick, a precise, and a relatively simple method of Oracle system contention identification. This increases the DBA's productivity and effectiveness by providing rock-solid problem identification. And as everyone knows, if you really know the problem, the resulting analysis and recommendations to solve the problem are much, much simpler.

While wait event analysis is a dramatic improvement or ratio based analysis, wait event analysis in itself is *incomplete*. As the figure below shows, wait event analysis focuses only on queue time, ignores service time, is concerned with isolating the Oracle bottleneck rather than determining response components, and quantifying user irritation. To encompasses the entire user response time experience, wait event analysis must be expanded into response time analysis.

---

1 I believe I was the first to publish a publicly available paper regarding Oracle wait event analysis in 1997. While I continue to update the paper, many others have since written about the value of using session wait event based performance analysis.
Figure 1. Response time is simply service time plus queue time. Ratio based analysis does not focus on service time or queue time and therefore impotent. Session wait event analysis focuses on queue time (i.e., Oracle wait related time). Oracle response time analysis expands on wait event analysis by including service time. This allows response time componentization plus allows user irritation definition.

Once again, the playing field is changing. Ratio based analysis led us to where a problem might reside and session wait event analysis told us specifically where the contention was and who was suffering. But a significant piece was still missing. Is the problem an issue? And if so, how much of an issue and what is the underlying problem? These final questions can not be answered using ratio based or wait event based analysis.

What is needed is a way to measure user irritation and then componentize the irritation cause to focus our performance tuning efforts.

What is needed is a way to measure response time and its components. Response time analysis takes us to another level of performance analysis. Response time analysis allows one to validate the real bottleneck (not just Oracle wait events), quantify user irritation (a measure of response time), and by componentizing response time one can optimally focus their performance effort.

For example, while profiling a query that really irritates a user, the following was discovered using the OraPub’s OSM-I tool, rtsess.sql (the actual output is shown and described in subsequent sections):

<table>
<thead>
<tr>
<th>Component</th>
<th>Time</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total response</td>
<td>7.22 sec</td>
<td>100%</td>
</tr>
<tr>
<td>Client proc</td>
<td>5.29 sec</td>
<td>73%</td>
</tr>
<tr>
<td>Oracle + O/S I/O</td>
<td>0.75 sec</td>
<td>10%</td>
</tr>
<tr>
<td>CPU proc time</td>
<td>1.18 sec</td>
<td>16%</td>
</tr>
</tbody>
</table>

This clearly indicates that we should focus our tuning efforts on client processing issues. There is probably some complex logic on the client side that needs to be optimized or perhaps a faster CPU is needed. This kind of information is absolutely key to quantitatively prove the performance problem is not server based. Even if we completely eliminated the Oracle related waits and the operating system I/O, we could only slice off 10% of the response time. This is just one simple example of how response time analysis can be used.

2 There are tools on the market today that do a superb job at profiling session activity. OraPub tools are free and therefore many time preferred to commercial tools. However, OraPub tools are typically not as professional looking and feature rich.
HOLISTIC PERFORMANCE ISOLATION METHOD (HPIM)

There are a number of ways, approaches, or methods to structure effective performance optimization. Because of past experiences, available training, job responsibilities, or even just plain "I don't want to go there" most Oracle DBAs focus their performance optimization on only the Oracle system. Regardless of whether the ratio based, wait event based, or response time based performance approach is used, this will always result in a partial and lop-sided solution when only the Oracle system is investigated.

A broader or holistic approach is to include the three key subsystems involved in every Oracle system [11,13]. These are the operating system, the Oracle system, and the application (system). By determining the contention in each subsystem and then observing the overlap, the problem quickly surfaces. This approach is very powerful because it is validated by three different, yet related, perspectives. A solid analysis using this holistic approach can not be broken.

DEFINING RESPONSE TIME

Response time is one of those words thrown around all the time yet there is little agreement on what it really means. What makes matters worse is that response time components are situation dependent. So for example, when you are referring to a queue time component someone else may be thinking you are referring to a service time component. All this to say, we need to understand response time definitions and basic queuing theory, to know how to represent response time (numerically and graphically), and finally understand how Oracle relates to response time.

INDUSTRY DEFINITIONS

The standard industry and mathematical response time related definitions are actually very simple. If you can relate these definitions to a real-life queuing situation (e.g., waiting in line at a fast-food restaurant) you will quickly gain a solid understanding.

- **Transaction.** A transaction is a unit of work. For example, getting money from an ATM or getting food from a fast-food restaurant.
- **Queue.** A queue is simply a line, a list, or a queue of transactions waiting to be serviced. This could be a fast-foot restaurant or an ATM line.
- **Queue Time.** This is how long a transaction waits or queues before it begins being serviced.
- **Server.** A server is simply a transaction processor. A CPU or a person at a fast-food counter is a good clean example of a server.
- **Service Time.** This is how long (e.g., seconds) it takes a server to service a transaction.
- **Response Time.** This is the summation of queue time (waiting in line) plus service time (being served).
- **Response Time Tolerance.** This is how much response time is acceptable. The higher the response time, the increase in user irritability.

INTRODUCTION TO QUEUING THEORY

Queuing theory and can be simple or very complex. For our purposes it needs to be very simple. First it is important to understand the above definitions. Once there is a solid definition grasp, how the pieces work together will become quickly clear.

In an incredibly simplistic (yet highly relevant) situation, a transaction enters a queue (queue time starts) and if the server is busy, waits. When a server completes servicing a transaction, it goes to the queue and removes the next queued transaction from the queue and begins servicing it. When our transaction is removed from the queue, queue time stops and service time begins. When the server completes servicing our transaction, service time stops and the server once again goes back to the queue for another transaction to service.

In reality, computing systems are composed of many queuing systems, sometimes called network queuing systems. But for our purposes we can keep things very simple while maintaining the required precision.
RESPONSE TIME REPRESENTATION

Understanding the definitions and understanding the basics of queuing theory allows one to then numerically and visually represent response time and then to also drill down into their components. To start this process, let’s begin with the basic numerical response time formula.

\[ R_t = Q_t + S_t \]

That is, response time \( R_t \) equals queue time \( Q_t \) plus service time \( S_t \). This classic response time graph is shown below.

![Response Time Graph](image)

**Figure 2. This is the classic response time graph.**

The response time curve, as we learned, is composed of service time plus queue time. Notice that when throughput is low, response time consists of only service time. As the number of transaction requests increase, that is, throughput increases, service time remains constant (by definition) but queue time will eventually begin to increase, resulting in an overall response time increase. As throughput continues to increase, response time reaches what is known as the knee in the curve where slight increases in throughput radically increases response time.

Looking at response time over a period of time provides us with a more realistic perspective of the system.
Figure 3. This graph shows response time information in its two basic components (service time and queue time) captured 23 times. Keep in mind that performance may not necessarily be bad during samples 8 thru 15.

It is important to recognize that while time samples 8 thru 15 may look horrible, the users may be perfectly content with performance.

The basic response time components only tell us where time is spent, not how painful that time spent may be.

What is needed is a way to measure user irritation. We must find a way to quantify how performance feels. This is not an easy task, but is essential to improve our performance analysis effectiveness. Later in this paper, I will discuss how this is done.

Drilling down into service time and queue time components depends on one’s perspective. That is, are we looking at response time from an operating system perspective or an Oracle perspective. Because perspective is absolutely key to continuing our discussion, the response time component discussion will be presented in another section below.

**ORACLE RESPONSE TIME MEASUREMENT CHALLENGES**

One would think because Oracle is a large player in the database industry, the capability to easily measure response time would be built into the product. Even with the word transaction thrown into just about every Oracle sales presentation, the concept of what really is a transaction and how to measure it is very challenging in today’s Oracle environment. There are four fundamental challenges with measuring, that is, quantifying Oracle response time. They are; no clear response time definition understood or accepted, no native Oracle transaction marker, CPU timing challenges, and ones perspective twists how response time is defined. Each of these is discussed below.

**NO CLEAR RESPONSE TIME DEFINITION**

This may seem rather silly at first, but the lack of a clear and accepted response time definition makes effective communication increasingly difficult. For example, what is service time, what is queue time, and what is wait time? Ask five people and you will probably receive three different answers. If one sticks with the fundamental, basic, and clear industry definitions, as presented in a previous section, response time definition becomes very clear with only one’s perspective to cloud the subject (more below).
**No Native Oracle Transaction Marker**

One would think because Oracle is a relational database system with the concept of a transaction fundamentally and mathematically established, the concept of response time would be a very simple extension. However, because Oracle does not naturally tag or mark a transaction, determining a transaction’s (whatever that is) service time, let alone its queue time, is very difficult. Because of this reality, one must be very clear when discussing Oracle response time. One must understand and explain if they are talking about a true transaction, a query, a DML operation, a specific session, a specific user, or an entire system. As one can quickly see, the absence of a transaction marker makes gathering, calculating, and presenting Oracle response time very challenging.

**CPU Timing Challenges**

There are three nasty CPU timing challenges we must be overcome. While these challenges are depressing at best, I have found correct contention identification, problem identification, and analysis unwavering (so far anyway). But please be cautious. Each of the challenges is quickly described below.

Oracle CPU reported timing can be just plain wrong. Many versions of Oracle do not correctly report CPU time (view `v$sysstat` and `v$sesstat` statistic name `CPU Used by this session`). To check if your system is suffering from this, run the OSM script, `timechk.sql`. It doesn’t take long to run, so run it multiple times on your system.

By default, Oracle only reports CPU usage when a process completes an operation. In the case of a long running PL/SQL process, CPU usage may only be reported until the outer-most block completes. To get around this problem, ensure the instance parameter `resource_limit` is set to `true`.

While Oracle has been working to improve this problem for years, chunks of time less than 100th of a second may not be recorded or reported by Oracle. This can significantly skew time based reporting. The more OLTP centric and the more small-chunks-of-activity centric your system is, the more skewed the results.

**Perspective Affects Definitions**

As with most things in life, perspective is significant. This is very true with response time in an Oracle based system. Ask yourself this, “Is a latch free wait queue time or service time?” or how about “Is a wait because an Oracle checkpoint is not completed queue time or service time?” Before you answer, consider looking at this question from three different perspectives: an operating system, an Oracle, and an application perspective.

- **Operating system perspective.** Suppose we consider O/S CPU time as service time. Therefore, everything else is lumped together as queue time. That is O/S I/O, Oracle latching, and application think time, for example. But from an O/S I/O perspective, I/O is service time and everything else is lumped together into queue time…including O/S CPU time. Are you beginning to see the problem?

- **Oracle database perspective.** From an Oracle database server perspective one typically labels service time as the time Oracle is processing a user’s request. But what does this really mean? Does this mean only CPU time related to a request is service time? What about the I/O time related to the request…is that service time? Are you beginning to see the problem?

- **Application user perspective.** This one is easy. Response time is how long it takes for control to be given back to the user. Nothing else matters! But while the definition is simple, this really doesn’t help the performance analyst much. Are you beginning to see the problem?

As you probably have figured out, one’s perspective makes all the difference when discussing response time. Unless I state otherwise, when I mention response time, I am coming from an Oracle perspective. This is discussed in more detail below.

**Measuring Oracle Response Time**

As discussed above, one’s response time perspective makes a tremendous difference where the performance issue resides. But is this really a problem? Surprisingly, this is usually not a problem. As long as the response time components can be identified and appropriately grouped and presented, whether the time is labeled as service time or as queue time is not a show stopper. What is relevant is where the time is spent and focusing on reducing where most of this time is spent. (This will be discussed in more detail below.)
**Idle Time and Think Time**

As you hopefully now appreciate, measuring Oracle response time is no small task. But to make matters worse, it is a quite a challenge to quantify and categorize user think time, presentation time, client side program time, network time, and lag time between when a user presses, for example, “commit” and when Oracle actually starts the commit.

Response time information is gathered either at the system level (the entire Oracle server) or for a specific session (i.e., Oracle session ID).

A complicating factor with gathering session level response time using Oracle performance views is we do not have a transaction marker. Therefore, we must start our timing clock, execute a transaction, and then stop our clock. While we try to minimize the time surrounding the actual transaction execution time, there will always be some time between when the clock starts and the transaction starts and also between when the transaction ends and the clock stops. I call this *idle time*. It always exists and is the result of measuring tool limitations and other factors.

Performing a “level 12 trace” and parcing the resulting trace file will eliminate timing error, but it can *not* account for presentation time and timing issues related to when a user presses the “commit” key and when Oracle actually starts the commit. While there are some very impressive commercial profiling tools available today which do eliminate the timing error, a “level 12 traces” does not provide a 100% end-to-end response time measurement. Also, timing error usually does not significantly affect ones analysis or conclusions.

When measuring response time at the session/transaction level, user *think time* does not exist. This is because think time occurs *between* transactions, not *during* a transaction.

From a system perspective, system idle time and user think time are closely related. In fact, there is no way to distinguish between the two. Therefore, when looking at response time from a system perspective, both system idle time and user think time are lumped into a single *idle time* category.

**Differing Response Time Needs**

A performance specialist must investigate a system from both an interactive and historical perspective. And in addition, one must look at specific processes and at the system as whole. When dealing with response time, if we understand the issues surrounding session level analysis and system level analysis, dealing with interactive and historical analysis seems to naturally make sense.

**Session Level**

When the topic of response time comes up, people generally are thinking at the session level. For example, one might say, “How long did it take *that* to run?” Because Oracle does not identify transactions like a transaction monitor, when we measure session level response time, there will always be some *left over* time we need to deal with.

Let me explain. Basically, we start the timer, the user runs *whatever* we are monitoring, when the *whatever* has completed, we stop the clock. There will always be a time gap between 1) when we start the timer and when the *whatever* begins and there will always be a time gap between 2) when the *whatever* ends and the timer stops. This is the *left over* time. While we try to minimize the *left over* time, at least a little bit will always exist.

Now the question is, “What to do with this *left over* time?” Keep in mind, if we could easily tag an Oracle transaction there would be no *left over* time, so we are trying make the best of an uncomfortable situation. Because the *left over* time is typically relatively small and because of its meaning, I call the *left over* time *idle time* and place *idle time* into *queue time*. 
Therefore, the Oracle session level response time analysis formulas are:

\[
\begin{align*}
R_t &= Q_t + S_t \\
S_t &= CPU_t \\
Q_t &= I/O_t + non-I/O_t + Idle_t
\end{align*}
\]

Where:

- \( R_t \) is response time
- \( Q_t \) is queue time
- \( S_t \) is service time
- \( I/O_t \) is I/O related queue time
- \( non-I/O_t \) is non-I/O related queue time
- \( Idle_t \) is the left over, that is, idle time

**SYSTEM LEVEL**

There will be times when knowing the overall system response time situation is very valuable. This allows one to quantify the overall user response time *irritability*. Looking at an Oracle system from a system level perspective is one of the best ways to determine if the bottleneck is worth the time to resolve.

Because response time is transaction based and because Oracle does not *mark* transactions, calculating response time at the system level is not possible. However, we can determine, for the system as a whole, where time has been spent. And that, as you will hopefully discover, is of great value.

*Idle time* takes on a new meaning when looking at things from an entire system perspective. *Idle time* is when the system is idle, that is, when it is waiting for something to do. This is easily measured within Oracle and occurs quite often. For example, when a user is thinking about something or when there is simply not enough work for the system to do, the system is idle.

A new term, called *Elapsed Time*, must now be defined. It is simply *response time* plus *idle time*. Another way of looking at this is how much time has been available since the system started. For example, if the system has been available for 10 minutes and there have been 5 users connected to the system the entire 10 minutes, the *elapsed time* is 50 minutes. (Unless parallel query or some other parallel feature is involved, a process’s queue time can not exceed wall time.) As more connections are made to the database system, *elapsed time* increases. In an Oracle system, *elapsed time* can not be easily collected, but because we know *idle time* and *response time*, we can derive *elapsed time*.

At a system level, the Oracle response time analysis formulas are defined slightly differently:
\begin{align*}
(5) & \quad E_t = R_t + \text{Idle}_t, \\
(6) & \quad R_t = Q_t + S_t, \\
(7) & \quad S_t = \text{CPU}, \\
(8) & \quad Q_t = \text{I/O} + \text{non-I/O},
\end{align*}

Where:

- $E_t$ is elapsed time
- $R_t$ is response time
- $Q_t$ is queue time
- $S_t$ is service time
- $\text{I/O}_t$ is I/O related queue time
- $\text{Non-I/O}_t$ is non-I/O related queue time
- $\text{Idle}_t$ is when the Oracle system is idle

When looking at response time interactively and at the system level, remember that unless you have a defined start and stop data gathering time and calculate the delta [13], you will be looking at response time since Oracle instance startup. If performance data is being gathered periodically, then the response time information will be much more useful.

The next section will explain how Oracle response time and its components are calculated in more detail.

**HOW TO MEASURE ORACLE RESPONSE TIME**

Because of the issues mentioned above, measuring Oracle response time can be challenging and somewhat complicated. However, it can be measured using only the Oracle virtual performance views (i.e., $v$S views). There are three steps involved. First, Oracle’s wait events must be categorized to properly identify service time and queue time. Second, we must identify and properly use our sources of data. As I mentioned above, we will use only Oracle’s performance views. And finally, a data gathering and reporting system [5] must be created to turn the data into useful information.

**CATEGORIZING WAIT EVENTS**

One of the main sources of data is Oracle’s wait events. Without the Oracle wait event interface, determining response time and its components would not be possible. But the situation becomes complicated because there are over 200 wait events currently defined and how to categorize the wait events can lead to some interesting conversations. Categorizing and standardizing the wait events is also important to ensure all related tools provide consistent information.

I categorize Oracle’s wait events into five areas:

- **I/O Read.** Time related to any Oracle process that waits for I/O read related information. This is considered queue time because an Oracle process is waiting for I/O because of an Oracle request to the operating system. An example wait event is `db file sequential read`.

- **I/O Write.** Time related to any Oracle process that waits for an I/O write to complete. This is considered queue time because an Oracle process is waiting for the I/O to complete as a result of an Oracle request to the operating system. Example wait events are `db file parallel write` and `log file checkpoint not complete`.

- **Idle.** Time related to when the computing system has more capacity to process what is being given to it to process. As mentioned above, the definition of idle time is dependent on the analysis type; session level or system level.

- **Bogus.** Time related to non-relevant wait events. I consider most wait events as `bogus` and the associated time is not included in any calculation. Since significant time is recorded for `bogus` wait events, this is an important category that must be carefully considered.
Other. Time related to non-I/O, idle, or bogus categories. The related time is considered queue time because an Oracle process is waiting for a section of kernel code to complete a task that could be completed faster if the O/S had more power. Discussing the wait events in this category typically result in very interesting conversations. Example wait events are latch free, log file sync, buffer busy waits, and enqueue waits.

Categorizing or componentizing response time allows the creation of a variety of very useful reports and graphs. Some of OraPub’s numeric reports [5] are shown in the following sections, but directly below are some of graphs based upon actual response time components.

Figure 5. This graph shows the basic response time components; service time and response time. Understand when response time increases, either there is simply more system activity, performance feels really bad, or both.

Figure 6. This graph details the basic Oracle queue time components. That is service time, I/O queue time (iow) and, non-I/O or other queue time or simply other time waited (otw).
Figure 7. This graph contains the exact same information as in the previous figure. What is different is how the information is displayed. This presentation reduces the emphasis on how bad performance might be and highlights where we need concentrate our performance tuning efforts.

Figure 8. This is an Oracle system level response time component drill down graphic. (The numbers are not related to the other graphs.) This is a good way to convey response time issues and performance strategies to others.

**Data Sources**

Only two basic views are needed to gather response time information. The session wait event views (`v$system_event`, `v$session_event`, `v$session_wait`) are used to gather queue time (I/O read, I/O write, other) and idle time (idle). The system statistic views (`v$sesstat`, `v$systat`) are used to gather service time (cpu time). System perspective idle time is gathered from the session wait views where the events are `sql*net message [from|to] [client|dblink]`. Service time is gathered from the session/system statistic views where the statistic name is `CPU used by this session`. 

Paper #515
**Turning Data into Useful Information**

While the sources of data and the basic formulas seem simple enough, properly gathering, consolidating, and appropriately reporting is entirely different. Shown below is a series of reports (interactive and historical) and graphs based upon the above sources of data and the basic response time formulas. As mentioned above, to fully investigate a system one must gather information both interactively (What’s going on now?) and historically (What’s been happening?). The first set of screen shots show interactive information, while the second set of screen shots show historically gathered information.

Another extremely important concept is user irritation. Gathering response time components is useful in itself because it shows us where time is spent. However, understanding queue time and service time does not quantify user irritation. What appears to be horrible response time does not imply users are dissatisfied with performance.

What is needed is a way to quantify not only response time, but user irritation. There are perhaps many ways to do this, but one of the best I have found is to either ask the user, “Are you irritated?” or to create a performance ratio composed of response time divided by elapsed time. I call this the Elapsed Time Response Time Ratio (ET RTR). The basic ET RTR formula is shown below.

\[
\text{ET RTR} = \frac{\text{Response Time}}{\text{Elapsed Time}}
\]

Based upon my personal experiences, I have found when the ET RTR exceeds 0.30 (i.e., 30%) and there is real application activity, users are generally dissatisfied with performance. When you review the system level response time reports in the figures below, remember to take a close look at the ET RTR.
Figure 9. This figure details response time for a specific session for a specific time frame. More specifically, this script was started (time starts ticking), the user ran their thing, the script was stopped (time stops ticking), and then this report was produced. This type of report is absolutely essential to truly understand why a process is taking so long.

```
DBA Server
The CPU time may be way off due to clock problems

--- Response Time Summary (RTSes)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Elapsed</th>
<th>Idle</th>
<th>Response</th>
<th>IO Wait</th>
<th>IO Read</th>
<th>Other Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>[cpu/resp]</td>
<td>[tv=resp]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/idle]</td>
</tr>
<tr>
<td>6.97</td>
<td>92.03</td>
<td>12.204</td>
<td>3</td>
<td>12.307</td>
<td>550</td>
<td>11.445</td>
</tr>
</tbody>
</table>
```

Figure 10. This figure details response time for the system as a whole since the Oracle instance was last started. This is a good way to get a general feel regarding response time and how the users are probably feeling about performance. Notice the Elapsed Time Response Time Ratio (ET RTR) is 1.000. This far exceeds 0.50 which means users are probably not pleased with performance.

```
DBA Server
The CPU time may be way off due to clock problems

--- Response Time Summary (RTSes)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Elapsed</th>
<th>Idle</th>
<th>Response</th>
<th>IO Wait</th>
<th>IO Read</th>
<th>Other Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>[cpu/resp]</td>
<td>[tv=resp]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/idle]</td>
</tr>
<tr>
<td>6.97</td>
<td>93.03</td>
<td>11.444</td>
<td>1.301</td>
<td>11.37</td>
<td>125</td>
<td>1.176</td>
</tr>
</tbody>
</table>
```

Figure 9. This figure details response time for a specific session for a specific time frame. More specifically, this script was started (time starts ticking), the user ran their thing, the script was stopped (time stops ticking), and then this report was produced. This type of report is absolutely essential to truly understand why a process is taking so long.

```
DBA Server
The CPU time may be way off due to clock problems

--- Response Time Summary (RTSes)

<table>
<thead>
<tr>
<th>CPU</th>
<th>Elapsed</th>
<th>Idle</th>
<th>Response</th>
<th>IO Wait</th>
<th>IO Read</th>
<th>Other Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>[cpu/resp]</td>
<td>[tv=resp]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/resp]</td>
<td>[tcpu/idle]</td>
<td>[tcpu/idle]</td>
</tr>
<tr>
<td>6.97</td>
<td>92.03</td>
<td>12.204</td>
<td>3</td>
<td>12.307</td>
<td>550</td>
<td>11.445</td>
</tr>
</tbody>
</table>
```

Figure 10. This figure details response time for the system as a whole since the Oracle instance was last started. This is a good way to get a general feel regarding response time and how the users are probably feeling about performance. Notice the Elapsed Time Response Time Ratio (ET RTR) is 1.000. This far exceeds 0.50 which means users are probably not pleased with performance.
Figure 11. This figure details response time for the system as a whole for the last 30 seconds as opposed to since the instance was started. This is a good way to get a general feel regarding response time and how the users are probably feeling about performance. Notice the Elapsed Time Response Time Ratio (ET RTR) is 1.000. This far exceeds 0.50 which means users are probably not pleased with performance. The tool used create this report is rtsysx.sql (OSM-I).
PERFORMING ORACLE RESPONSE TIME ANALYSIS

If you have a good grasp of the information presented so far in this paper, then how to use and apply the information should come relatively easy. This very brief section assumes you understand the above sections and will demonstrate how to use the information presented previously in this paper.

HOW TO MEASURE RESPONSE TIME.

Response time is simply how long it takes to do something. Classically this is service time plus queue time. In an Oracle environment, two additional measurements are needed. These are idle time and elapsed time. I define these terms in an Oracle environment as follows.

- **Idle time** is either timing error (session level only), user think time (system level only), network transport time (both session and system), presentation time (both session and system), and the time lag between when a user executes, for example, a “commit” and when Oracle actually begins the commit (both session and system).
- **Service time** is the CPU time.
- **Queue time** is I/O related waits and other non-I/O related waits (e.g., enqueue, latch, buffer busy). *Idle time* is included in *queue time* only during session level analysis.
- **Response time** is *service time plus queue time*.
- **Elapsed time** is *response time plus idle time*.

The sources of data are Oracle’s session wait views (`v$session_wait`, `v$session_event`, `v$system_event`) and the basic statistic views (`v$sesstat`, `v$sysstat`). By cleverly gathering data from these data sources, one can categorize time into service, idle, queue, and elapsed time. That is, we can measure response time and user irritation.

Quantifying response time simply tells us where time has been spent. It does not tell us if a user is irritated by the response time. To measure user irritation I use a simple performance ratio composed of response time divided by elapsed time, or ET RTR.

Figure 12. This report summarizes response time during (i.e., within) each of the data gathering periods. This type of report is essential to fully understand the system’s response time characteristics. Notice the first column, *Elapsed Time Response Time Ratio* (ET RTR), numbers. When this performance ratio exceeds 0.30 (i.e., 30%) and there is real application activity, users tend to feel performance is not acceptable.
OraPub’s System Monitor tool kit [5] or OSM for short, has a number of response time related reports. The OSM tool kit also covers both interactive (session and system perspective) and historical requirements. Historical reports come in two flavors: accumulated and delta [11,13]. Accumulated reports show the data since the Oracle instance has started and delta reports show information within specific time periods. Delta reports are more complicated to develop but far more useful then accumulated reports. In the historical tool listing below, the “a” reports are accumulation based and the “d” reports are delta based. Some examples of these reports have been shown previously in this paper. Below are the actual tool names along with a short description.

rtsum. Interactive system level response time summary report.
rtio. Interactive system level I/O wait summary wait event details.
rtow. Interactive system level non-I/O wait summary with wait event details.
rtsys. Simply runs rtsum, rtio, and rtow.
rtss [a,d]. Interactive session specific response time details.
rtia [a,d]. Historical system level response time summary report.
rtio [a,d]. Historical system level I/O wait summary report.
rtiew [a,d]. Historical system level I/O wait event detail report.
rtow [a,d]. Historical system level non-I/O wait summary report.
rtowe [a,d]. Historical system level non-I/O wait event detail report.
timechk <interval>. Checks Oracle CPU time reporting.

HOW TO VALIDATE A BOTTLENECK
Using classic ratio based or even with session wait based analysis, the true bottleneck can in some cases be mistaken. Session wait based analysis tells us where in the Oracle kernel code processes are waiting, but session wait based analysis does not tell us anything about service time. Response time analysis takes session wait based analysis to the next level by gathering service time and componentizing response time related time.

A simple way to validate the bottleneck is to look at the ratio of either service time to response time or the ratio of queue time to response time. (These two ratios added together should always equal 1.00.) If the service time to response time ratio is greater than 50%, then service time is the main bottleneck. If the service time to response time ratio is less than 50%, then we would look at the queue time components. Reference the reports previously shown in the paper and note these ratios.

HOW TO QUANTIFY USER IRRITATION
As mentioned previously (many times), response time tells us where time was spent, but does not tell us if users are satisfied with performance. What is needed is a measure of irritation. A simple ratio consisting of response time divided by elapsed time, named the Elapsed Time Response Time Ratio (ET RTR) can be used. Experience shows users are generally irritated with performance when the ET RTR is above 30% and there is real application activity. Reference the reports previously shown in this paper and note this ratio—or better yet run these scripts on your own system.

HOW TO FOCUS PERFORMANCE TUNING EFFORTS
Once the bottleneck has been validated and we know users are irritated, it is time to either find a new job or solve the problem. Using response time analysis, we can confidently, effectively, and very efficiently focus our performance tuning efforts.

Here is how to do it. Assuming the bottleneck has been validated and users are irritated, determine whether to focus on either service time or queue time. If the service time to response time ratio is greater than 50%, then focus on service time (i.e., tuning the SQL, increase CPU power, etc.). If queue time is the issue, then break down the queue time into its components and focus on the area that consumes the most queue time. That may also result in tuning the SQL, but it will undoubtedly focus on other issues as well (e.g., increasing I/O capacity, reducing I/O requirements, use bind variables, etc.).
CASE STUDIES

Grasping new concepts is difficult. Especially when one has been trained with a different mindset or way of doing things. Therefore, I believe case studies and hands-on exercises are critical before head knowledge can be turned into productive work. Three distinct case studies are below, each detailing response time analysis from a different and relevant perspective.

INTERACTIVE SESSION LEVEL PERSPECTIVE

This is a very simply case study but will provide a method for you to being applying response time analysis. A specific user is extremely irritated with performance. In fact, the problem/irritation is centered on a specific query. As a result, you had the user run the query while you where gathering response time information (using the OSM tool, rtsess.sql <sid>). The report looks like this:

We know there is a bottleneck (because there always is a bottleneck) and we know the user is irritated (they told us that). We have decided not to quit our job, so we must determine where to focus our tuning efforts. The above report’s Response Time Summary shows the service time to response time ratio is 6.88%, which means 93.12% of the response time is related to queue time. We obviously will focus on the queue time. The above report’s Queue Time Summary shows that of the 4.33 seconds of queue time, 3.41 seconds is related to I/O waits. Therefore, we will concentrate our efforts on I/O. The Queue Time I/O Timing Detail shows that the key problem is processes waiting for I/O from a full-table scan (wait event db file scattered read). Now we perform classic session wait analysis to determine which table(s) is involved, which SQL statement(s) is involved, and ultimately and gloriously solve the problem.
This is a very simply case study but will provide a method for you to being applying response time analysis. We are told to simply take a look at the system. We don’t know if users are pleased with system performance, we just want to take a look and see if there is anything we can do. Not entirely realistic I know, but you’ll get the idea. Your analysis begins with running the OSM’s $\texttt{rtsys.sql}$ tool. This particular report looks at the system since the Oracle instance started. The results are shown below.

We know there is a bottleneck (because there always is a bottleneck) but we don’t know how irritated the users may be. By looking at the report’s Response Time Ratios section, we see the Elapsed Time Response Time Ratio (ET RTR) is 1.000. This far exceeds our general rule of thumb of 30%, so we believe users are fairly dissatisfied with system performance. To determine where to focus our efforts we first look at the Response Time Summary where we see the service time to response time ratio is 6.97%, meaning 93.03% of the time users are waiting because of queuing issues as opposed to CPU issues. We also notice that the majority of queue/wait time is related to non-I/O issues (i.e., 10,143 minutes compared to 1,301 minutes). Therefore, we look at the Response Time Other Waits (non-I/O) Event Detail where we find that the majority of wait time is related to the buffer busy wait event. To validate this bottleneck, we need to investigate the recent response time activity, not just since the instance has started. The next case study does just this.

This case study begins where the previous case study ended. We began our analysis began by looking at the system since the instance has started using the OSM $\texttt{rtsysx.sql}$ tool. However, what has occurred since instance startup can be very different then what has recently occurred. What is needed is information about what has happened over the last, let’s say, 60 seconds. The OSM tool to view recent response time activity at the system level is $\texttt{rtsysx.sql}$. The results are shown below.
We know there is a bottleneck (because there always is a bottleneck) but we don’t know how irritated the users may be. By looking at the report’s Response Time Ratios section, we see the Elapsed Time Response Time Ratio (ET RTR) is 1.000. This far exceeds our general rule of thumb of 50%, so we intensely believe users are extremely dissatisfied with system performance.

To determine where to focus our efforts we first look at the Response Time Summary where we see the service time to response time ratio is 13%, meaning 87% of the time users are waiting because of queuing issues as opposed to CPU issues. We also notice that the majority of queue/wait time is related to non-I/O issues (i.e., 378 seconds compared to 49 seconds). Therefore, we look at the Response Time Other Waits (non-I/O) Event Detail section where we find that the majority of wait time is related to the buffer busy waits. The next step is to perform a classic session wait event analysis focusing on the buffer busy wait event.

**HISTORICAL SYSTEM PERSPECTIVE**

This is a very simply case study but will provide a method for you to being applying response time analysis. We are told to simply take a look at the system. However, this analysis is based upon historical data, not interactive. Normally, one would look at the system from both an interactive and a historical perspective, but for this case study we are only looking from a historical perspective. We don’t know if users are pleased with system performance, we just want to take a look and see if there is anything we can do. Not entirely realistic I know, but it works for our purposes. We installed the OSM tool kit, gathered performance data, and ran the response time summary report rtsd.sql, which is shown below.
3. Determine from both an operating system and an application perspective where the contention/high-usage/bottlenecks are, specifically on I/O. There are other OSM reports that can be run to detail the I/O issues. And of course we would want to wait time is around 1500 minutes with around 1500 of those minutes related to I/O wait, we know to focus our efforts. With the average response time ratio is only around 19%, meaning around 81% of the response time is related to queuing/wait issues. Because the total company takes an early lunch. To determine whether we focus on service time or queue time, we notice the cpu to response time peaks at 0.964 around lunch time (time key 55 and 56). After checking with the users, we find out that around lunchtime is when things painfully slow down. Unusual for most companies, but this is common for companies that offer an early lunch. We know there is a bottleneck (because there always is a bottleneck) but we don’t know how irritated the users may be. Notice the Elapsed Time Response Time Ratio (i.e., RT/ET) at peaks at 0.964 around lunch time (time key 55 and 56). After checking with the users, we find out that around lunchtime is when things painfully slow down. Unusual for most companies, but this company takes an early lunch. To determine whether we focus on service time or queue time, we notice the cpu to response time ratio is only around 19%, meaning around 81% of the response time is related to queuing/wait issues. Because the total wait time is around 1500 minutes with around 1500 of those minutes related to I/O wait, we know to focus our efforts specifically on I/O. There are other OSM reports that can be run to detail the I/O issues. And of course we would want to determine from both an operating system and an application perspective where the contention/high-usage/bottlenecks are.

CONCLUSION

This paper was written for those who want to push the Oracle system performance analysis envelope. Oracle performance tuning has evolved over the years and my hope is that we will all be flexible and open to new performance optimizing techniques regardless of our past, our experiences, our peer group, and the various economic incentives that surround us. If you have questions or comments, please feel free to email me. My objective is to further the art of Oracle performance optimization and open discussion is a great way to further this objective. Thank you for your valuable time.

REFERENCES


**ACKNOWLEDGMENTS**

A special thanks to my email acquaintance, clients, and students who have brought forth a plethora of stimulating discussions and challenging dilemmas. These situations coupled with my unusual enthusiasm to Oracle performance analysis has evolved into this technical paper.