Response Time Analysis For Oracle Based Systems

The next Oracle performance management frontier

Latest version available at http://www.orapub.com
This page has been intentionally left blank.
Table of Contents

TABLE OF CONTENTS ................................................................................................................. 3

1. INTRODUCTION .................................................................................................................. 4
   1.1 EVOLUTION OF ORACLE PERFORMANCE ANALYSIS ................................................ 5
   1.2 HOLISTIC PERFORMANCE ISOLATION METHOD (HPIM) ............................................ 8

2. DEFINING RESPONSE TIME ............................................................................................... 8
   2.1 INDUSTRY DEFINITIONS ............................................................................................... 8
   2.2 INTRODUCTION TO QUEUING THEORY ..................................................................... 9
   2.3 RESPONSE TIME REPRESENTATION .......................................................................... 10
   2.4 ORACLE RESPONSE TIME MEASUREMENT CHALLENGES ..................................... 11
       2.4.1 No clear response time definition ........................................................................ 12
       2.4.2 No native Oracle transaction marker .................................................................... 12
       2.4.3 CPU timing challenges ......................................................................................... 12
       2.4.4 Perspective affects definitions ............................................................................. 13

3. MEASURING ORACLE RESPONSE TIME ........................................................................... 13
   3.1 IDLE TIME AND THINK TIME ....................................................................................... 14
   3.2 DIFFERING RESPONSE TIME NEEDS ......................................................................... 14
       3.2.1 Session Level ......................................................................................................... 14
       3.2.2 System level ........................................................................................................... 16
   3.3 HOW TO MEASURE ORACLE RESPONSE TIME ....................................................... 17
       3.3.1 Categorizing wait events ...................................................................................... 17
       3.3.2 Data Sources ......................................................................................................... 20
       3.3.3 Turning data into useful information ...................................................................... 20

4. QUANTIFYING USER IRRITATION ...................................................................................... 24

5. VALIDATING A BOTTLENECK ............................................................................................. 25

6. ORAPUB’S RESPONSE TIME ANALYSIS TOOLS .............................................................. 26

7. CASE STUDIES .................................................................................................................. 27
   7.1 INTERACTIVE SESSION LEVEL PERSPECTIVE ......................................................... 27
   7.2 INTERACTIVE SYSTEM PERSPECTIVE #1 ................................................................. 28
   7.3 INTERACTIVE SYSTEM PERSPECTIVE #2 ................................................................. 29
   7.4 HISTORICAL SYSTEM PERSPECTIVE ....................................................................... 30

8. CONCLUSION ..................................................................................................................... 32

9. REFERENCES ...................................................................................................................... 33

10. ACKNOWLEDGMENTS ..................................................................................................... 33

11. ABOUT THE AUTHOR ...................................................................................................... 33
Response Time Analysis

for

Oracle Based Systems

Craig A. Shallahamer (craig@orapub.com)

Original Sept. 3, 2001

Version 6b, January 26, 2004

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 it 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 makes no distinction between a database transaction and a business transaction, 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.

1. 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 was 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.
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.

1.1 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 and, 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 requesting a resource (e.g., latch, physical i/o, cached block, enqueue, etc.) but that resource 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 its 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 over ratio based analysis, wait event analysis is in itself incomplete. As the figure below shows, wait event analysis focuses primarily on queue time, for the most part it ignores service time, and it is more concerned with isolating the Oracle bottleneck rather than determining response time components and quantifying user irritation. To encompass more of the user response time experience, wait event analysis must be expanded into Oracle 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.
Oracle Response Time 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 component categorization 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 Oracle 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 categorize 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 categorizing response time components one can optimally focus their performance effort.
For example, while profiling a query that really irritated a user, the following was discovered using OraPub’s interactive session profiling tool, \texttt{rtsess.sql} (the actual output is shown and described in subsequent sections):

<table>
<thead>
<tr>
<th>Component</th>
<th>Time</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total response</td>
<td>7.22 sec</td>
<td>100%</td>
</tr>
<tr>
<td>Network + Client proc</td>
<td>5.29 sec</td>
<td>73%</td>
</tr>
<tr>
<td>I/O (Oracle + O/S)</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.

### 1.2 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 \cite{1,2,11}. 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.

### 2. 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.

#### 2.1 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.

\footnote{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.}
• **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 (in American English), a list, or a queue of transactions waiting to be serviced. This could be a fast-foot restaurant or a cash machine line.

• **Queue Time.** This is how long a transaction waits or queues before it begins being serviced. The is an unlimited amount of available queue time. That is, a transaction can be waiting in a queue forever.

• **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. In contrast to queue time, there is a fixed amount of available service time. For example, a single CPU system has only 60 minutes of available CPU service time each hour. Understanding that available service time is fixed yet queue time is unlimited is key to understanding a measuring user irritation.

• **Response Time.** This is the sum of queue time (waiting in line/queue) 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.

### 2.2 Introduction to queuing theory

Queuing theory 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.
2.3 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 its 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.

![Figure 2. This is the classic response time graph.](image)

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 or the elbow 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.
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.

2.4 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 is 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.
2.4.1 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).

2.4.2 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. Unfortunately this is not the case. There is a significant difference between a business transaction and a native Oracle transaction. Users think in terms of a business transaction and therefore response time must be business transaction based. Because Oracle does not naturally instrument, tag, or mark a business 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 what kind of transaction they are referring to and if they are looking at the situation from a specific session, a specific user, or an entire system perspective. As one can quickly see, the absence of a business transaction marker makes gathering, calculating, and presenting Oracle response time very challenging.

2.4.3 CPU timing challenges

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

Many versions of Oracle do not correctly report CPU time. The statistic CPU Used by this session is typically what is used to gather CPU time. However, there can be serious issues. For this statistic, the system level v$sysstat view commonly reports wildly incorrect CPU times. The session level v$sesstat view typically shows the correct CPU time. A better solution is to use the OS User time used statistic. (Make sure you set the instance parameter timed_os_statistics to a value greater than 0.) This usually produces correct values. You can use the OSM script, timcheck.sql, to do a quick timing check on your specific session. It doesn’t take long to run, so run it multiple times on your system.

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

Another challenge is understanding where Oracle records the various CPU time components. The statistic CPU Used by this session includes only CPU user time, not CPU system time. However the relatively new statistics OS user time and OS system time contain, as you might guess, CPU

---

3 I have not personally verified this.
user time and CPU system time. Because Oracle’s wait events are actually O/S system calls, they can contain CPU system time. This is especially true when the system time to user time ratio increases. Keep this in mind as I describe the fundamental response time formulas below.

There are other challenges, but for system or session level response time analysis, understanding and dealing with these issues should provide you with precise enough values for advanced diagnostics.

2.4.4 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 data transfer time is service time and rotational delay and head movement time is queue time. Are you beginning to see the problem?

* Oracle database perspective. From an Oracle database server perspective one typically labels service time as Oracle kernel execution time. 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? How about time spent trying to get a latch…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.

3. 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.)

---

4 Answer: Yes, when kernel code is repeatedly requesting (i.e., spinning) for a latch. No, when the user process goes to sleep for a fixed amount of time before spinning once again.
### 3.1 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 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 place this time in the **unaccounted for time** bucket. It always exists and is the result of measuring tool limitations and other factors.

Performing a “level 12 trace” and parsing the resulting trace file can help reduce this 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 minimize the timing error, a “level 12 traces” does not provide a 100% end-to-end response time measurement.

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.

### 3.2 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 a 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.

#### 3.2.1 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. There will also be some time Oracle does not keep track of, that is, **uninstrumented time**. Hopefully this **uninstrumented time** is negligible. I place all **left over** time and **uninstrumented time** into a bucket named **unaccounted for time**. When performing a session level response time analysis, you want to minimize **unaccounted for time**.

Let me explain timing error in a little more detail. Unless you have instrumented your application (or at least the transaction being analyzed), there will always be some extra time. Basically, we start the timer, the user runs whatever we are monitoring, and 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
Oracle Response Time Analysis

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.

It may be possible to instrument your application or perhaps the key application transactions. Monitoring your key business transactions both interactively and historically gives us a performance management advantage.

**Server Side Happenings**

<table>
<thead>
<tr>
<th>CPU used by this session</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL*Net message from client</td>
</tr>
<tr>
<td>db file scattered read</td>
</tr>
<tr>
<td>db file sequential read</td>
</tr>
<tr>
<td>log file sync</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Start</td>
</tr>
<tr>
<td>Real Response Time</td>
</tr>
<tr>
<td>Start Clock</td>
</tr>
<tr>
<td>Recorded Response Time</td>
</tr>
<tr>
<td>Query End</td>
</tr>
<tr>
<td>Stop Clock</td>
</tr>
</tbody>
</table>

![Figure 4](image)

**Figure 4.** This chart shows one way to represent session level response time and its components. This chart highlights the time gap between clock start and query start and between query end and clock stop. This chart also highlights how various response time components are combined to determine response time.

Therefore, the Oracle perspective session level response time analysis formulas are:

\[
(1) \quad R_t = Q_t + S_t + UA_t
\]

\[
(2) \quad S_t = CPU_t
\]

\[
(3) \quad Q_t = I/O_t + NC_t + Other_t
\]

Where:

- \( R_t \) is response time
- \( Q_t \) is queue time, which DBAs look at as non-CPU time.
- \( S_t \) is service time, which DBAs look at as CPU time.
- \( UA_t \) is unaccounted for time which is composed of tools usage error (i.e., timing error), uninstrumented Oracle kernel code time, and Oracle time accounting error
- \( I/O_t \) is physical I/O related wait/queue time.

\(^5\) I say, “Oracle perspective” because service time is really only CPU *user* time and the CPU *system* time is accounted for in the queue time (i.e., wait event time) bucket.

\(^6\) Oracle says its physical, but the data could also be in a file system buffer cache or an IO subsystem cache.
Other is various non-IO related wait/queue event time.

NC is the network delay and client program processing wait/queue time.

3.2.2 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 business transaction based and because Oracle does not instrument business transactions, calculating transaction level 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 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.

It is a fact, that a 6 CPU system has a fixed amount of 21600 seconds of available CPU seconds each hour (6 CPUs X 60 seconds/minute X 60 minutes/hour). How much CPU is being used or consumed can be represented as a percentage of CPU used divided by CPU available. This is commonly called CPU utilization. For example, if 3000 seconds of CPU are being used, yet there is 21600 seconds of available CPU time, then CPU utilization is 0.139 or around 14%. We know from both queuing theory and observation that queue time consumes around 10% of response time even though CPU utilization is only at around 70%. That can be very unsettling if you are used to running your system at around 90% utilization. (By the way, at 90% utilization, queue time accounts for over 46% of response time!)

In contrast to service time, queue time is unconstrained. That is, queue time is not a resource or something to be consumed. It’s simply what you do when you are not being served. From an Oracle perspective, queue time is composed of CPU queue time, O/S I/O service and queue time, and other Oracle related queue time. So the situation is a more complicated and not as clean as most people suspect.

Therefore, queue time will continue to increase as the system load increases, while CPU service will reach a maximum which can not be exceeded. It is important to understand that because queue time is unconstrained, both response time and therefore, elapsed time are also unconstrained and have no maximum value.
At a system level, the Oracle response time analysis formulas are defined slightly differently:

\[
R_t = Q_t + S_t
\]

\[
S_t = \text{CPU}_t
\]

\[
Q_t = \text{I/O}_t + \text{non-I/O}_t
\]

Where:

- \(R_t\) is response time
- \(Q_t\) is queue time
- \(S_t\) is service time. This is actually only CPU user time. CPU system time is not unaccounted for because it is included in the Oracle wait event time (i.e., \(Q_t\)).
- \(\text{I/O}_t\) 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 based computing 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 \([9,11]\), you will be looking at response time since the Oracle instance startup perspective. 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.

### 3.3 How to measure Oracle response time

Because of the issues mentioned above, measuring Oracle response time can be challenging, inaccurate, and somewhat complicated. However, it can be measured using only the Oracle virtual performance views (i.e., \(v\$\) 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 [6] must be created to turn the data into useful information.

#### 3.3.1 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 around 1000 wait events currently defined (Oracle 10g) 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`.

- **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.

- **NC.** Time related to network and client side processing. The `SQL*Net message` related wait events are all considered NC events.

- **Other.** Time related to non-I/O, non-NC, and non-non-bogus categories. Example wait events are `latch free`, `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 [6] are shown in the following sections, but directly below are some of graphs based upon actual response time components.

![Oracle System Response Time](image)

**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.
3.3.2 Data Sources

Only two basic views are needed to gather response time information. The session wait event views (v$system_event, v$session_event) are used to gather queue time. The system statistic views (v$sesstat, v$sysstat) are used to gather service time (i.e., CPU time). The statistic is OS User time used. (CPU system time is included in wait event time.)

3.3.3 Turning data into useful information

While the sources of data and the basic formulas may 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. I regularly update my tools, so when you run the tools (e.g., rtsess.sql), the output may look slightly different. 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.
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.
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 response time feel regarding how the users are probably feeling about performance.
Figure 11. This figure details response time for the system as a whole for the last 120 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. The tool used create this report is rtsysx.sql (OSM-I).
4. Quantifying User Irritation

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 besides asking a user, “Are you irritated?”, combining two different ratios is proving (not proved, but proving) to be a good user irritant indicator.

Users are irritated because of poor response time. Response time is composed of queue time and service time. If a performance ratio is created centered around CPU time and another performance ratio is created centered around queue time, and then if either ratio becomes large, we know there is probably a performance problem. The individual ratios and their combination is shown below.

\[
\text{Response Time Ratio} = \text{Maximum (CPU Busy Ratio, Queue Time Ratio)}
\]

Where:

\[
\text{CPU Busy Ratio} = \frac{\text{Service Time}}{\text{Available CPU Time}}
\]

\[
\text{Queue Time Ratio} = \frac{\text{Queue Time}}{\text{Response Time}}
\]

Both the CPU Busy Ratio and the Queue Time Ratio will be between 0 and 1. (If Oracle is reporting bogus system CPU time, the CPU Busy Ratio can be much larger than one.) By taking the maximum of these two ratios, the overall Response Time Ratio can indicate a performance
problem. Be aware that this ratio is only meaningful when there is real work occurring in the database system. It is very common for a near 1.0 value to occur when little work is or has been performed.

After observing the Response Time Ratio (RTR) in many different Oracle production situations combined with a basic knowledge of queuing theory, anytime the RTR exceeds around 75% and there is significant work occurring in the database system, users tend to be irritated. I suspect this ratio will be modified in the future, but for the time being a “RTR 75% rule” seems to work well.

5. Validating 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 (server and background) 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 categorizing response time related components.

At first glance, one would think simply looking at the service time to response time ratio would indicate if the issue is service time related or queue time related. Unfortunately, it’s not that simple. Every computer system has a fixed amount of CPU time and an unlimited potential of queue time. Therefore, a CPU bottlenecked system with lots of wait event time could look like a queuing issue when really the main bottleneck is a CPU shortage.

Let me give a common example. When there is significant latching contention server processes are both consuming CPU (spinning to get a latch) and waiting (sleeping for a fixed short time). On a latch bottlenecked system, the CPUs subsystem will probably be bottlenecked…so you could say there is a CPU power shortage. However, significant latch contention means processes are sleeping in addition to spinning. The sleep time is recorded as an Oracle wait event. If enough process are sleeping, wait time will far exceed service time. So one may think the bottleneck is latch contention…and it is from an Oracle perspective, but from an operating system perspective it is clearly a CPU bottleneck.
To properly validate the bottleneck, we need, at a minimum, both the operating system CPU utilization and whether the wait events point towards an IO or CPU bottleneck. Once we have this information, we can pinpoint the bottleneck, which is roughly shown in a matrix below. Use this matrix as a guideline and to facilitate discussion to arrive at the true bottleneck, the cause of the bottleneck, and alternative solutions.

<table>
<thead>
<tr>
<th>Wait Events</th>
<th>Wait Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IO Centric</td>
</tr>
<tr>
<td>CPU Utilization HIGH</td>
<td>IO controller, file system buffer cache, etc. issues</td>
</tr>
<tr>
<td>CPU Utilization LOW</td>
<td>IO Bottleneck</td>
</tr>
</tbody>
</table>

This is a lot more there can be said about using response time analysis to isolate and confirm the operating system bottleneck, but that is out of scope for this paper. However, I discuss this topic in my Advanced Reactive Performance Management class [1].

6. OraPub’s Response Time Analysis Tools

OraPub has a free performance centric tool kit anyone can download and use for free from OraPub’s web site (http://www.orapub.com/tools). The OraPub System Monitor tool kit [6] 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 reporting 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.
- **rtsess <session id>.** Interactive session specific response time details.
- **rts[a,d].** Historical system level response time summary report.
- **rtio[a,d].** Historical system level I/O wait summary report.
- **rtoe[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.
Oracle Response Time Analysis

timechk <interval>. Checks Oracle CPU time reporting.

7. 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 shown below, each detailing response time analysis from a different and relevant perspective.

7.1 Interactive Session Level Perspective

This is a very simple 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, rtssess.sql <sid>). The report looks like this:

```
192.168.0.15 - PuTTY

Session level response time details for SID 14

*** Response Time Summary

<table>
<thead>
<tr>
<th>Time(sec)</th>
<th>Service</th>
<th>Queue Time</th>
<th>% CPU</th>
<th>% Queu</th>
<th>% Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>[rt+st+qt]</td>
<td>[st]</td>
<td>[qt]</td>
<td>[uqt]</td>
<td>[st/rt]</td>
<td>[qt/rt]</td>
</tr>
<tr>
<td>45.17</td>
<td>5.34</td>
<td>34.61</td>
<td>5.22</td>
<td>11.62</td>
<td>76.62</td>
</tr>
</tbody>
</table>

*** Queue Time Summary

<table>
<thead>
<tr>
<th>Queue Time</th>
<th>I/O(sec)</th>
<th>Net+Client(sec)</th>
<th>Other(sec)</th>
<th>QT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[qio+nqo+qt]</td>
<td>[qio]</td>
<td>[nqo]</td>
<td>[qo]</td>
<td>[qt]</td>
</tr>
<tr>
<td>34.61</td>
<td>34.59</td>
<td>0.92</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

*** Queue Time IO Timing Detail

<table>
<thead>
<tr>
<th>I/O(sec)</th>
<th>QT</th>
<th>Read I/O(sec)</th>
<th>Write I/O(sec)</th>
<th>% Writes Time</th>
<th>% Read Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[prior+write]</td>
<td>[qio]</td>
<td>[rrio]</td>
<td>[wrio]</td>
<td>[wrio*rio]</td>
<td>[read*rio]</td>
</tr>
<tr>
<td>34.59</td>
<td>23.50</td>
<td>1.09</td>
<td>96.86</td>
<td>3.15</td>
<td></td>
</tr>
</tbody>
</table>

*** Queue Time IO Event Timing Detail

<table>
<thead>
<tr>
<th>Wait Event Name</th>
<th>Wait Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>db file sequential read</td>
<td>1.06</td>
</tr>
<tr>
<td>log file switch (checkpoint incomplete)</td>
<td>33.12</td>
</tr>
<tr>
<td>log file switch completion</td>
<td>0.86</td>
</tr>
<tr>
<td>log file sync</td>
<td>0.82</td>
</tr>
</tbody>
</table>

*** Queue Time Other Event Timing Detail

*** Wait Event Time Not Categorized (for QA)

<table>
<thead>
<tr>
<th>Wait Event Name</th>
<th>Wait Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhbas ipc reply</td>
<td>0.94</td>
</tr>
</tbody>
</table>
```
We know there is a bottleneck (because there always is a bottleneck) and we know the user is irritated (they clearly told us that). We have decided not to quit our job or shamelessly seek a promotion, 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 12% and the majority of time is spent as queue time (77%). An unaccounted for time of 12% of response time is OK. We checked, and there is plenty of CPU time. Clearly we should focus on the queue time. The above report’s Queue Time Summary shows that of the 34.51 seconds of queue time, 34.59 seconds are 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 to complete writes to the IO subsystem (97%). Specifically, the session is waiting for the DBWR to finishing writing dirty buffers to “dbf” files.

We can now work on the problem from a number of different perspectives. From an Oracle perspective, we could increase the redo log file size to reduce the checkpoint frequency. From an application perspective, we would try to understand and reduce the amount of redo being generated. From an operating system perspective, we would increase IO writing capacity specifically for the DBF files typically involved. As you can see, there are always multiple approaches to solve the same problem.

7.2 Interactive System Perspective #1

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 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. While the Response Time Ratio is not shown in this screen shot, it is clearly above our 75% rule of thumb. Next we must determine where to focus our efforts. We first look at the Response Time Summary. Here we see the service time to response time ratio is 6.97%. This means, from an Oracle perspective, that 93.03% of the time users are waiting because of queuing issues as opposed to CPU issues. However, from an operating system perspective, we checked and there is clearly available CPU power. 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 section 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.

### 7.3 Interactive System Perspective #2

This case study begins where the previous case study ended. We began our analysis by looking at the system since the instance has started using the OSM `rtsys.sql` tool. However, what has occurred since instance startup can be very different than 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 `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. While the **Response Time Ratio** is not shown in this screen shot, it is clearly above our 75% rule of thumb, indicating 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%. This means, from an Oracle perspective, 87% of the time users are waiting because of queuing issues. We checked at the operating system level and there is also plenty of available CPU time. 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.

### 7.4 Historical System Perspective

This is a very simply case study but will provide a method for you to begin 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.

We know there is a bottleneck (because there always is a bottleneck) but we don’t know how irritated the users may be. The Response Time Ratio is above our 75% rule of thumb around midnight, which is, after checking with the users, when batch performance is painfully slow. To determine where we focus our tuning efforts, we look at the Oracle CPU time versus the Oracle IO wait time. Notice that the CPU time does not increase during peak time but IO wait time dramatically increases. We checked at the OS level and it also indicates an IO bottleneck. This is a clear indication we need to focus our efforts specifically on IO. To get more Oracle IO specifics, we run the historical rtiod.sql (OSM-H) report. It is shown below.
The report clearly shows Oracle is waiting predominately for Oracle to finish writing (this specific report does not tell us whether it is a data file or a redo log issue). However, the read waiting is also significant.

Therefore, we look at SQL that is consuming a lot of physical IO (writing: DML, reading: SELECT). To decrease the read related waits, among other things, we look at Oracle’s buffer cache and see if it can be responsibly increased. We also look at the IO subsystem and investigate how we can increase both write and read capacity. This could be as simple as moving Oracle database files around. But our solution should be approached from all three perspectives. This give us a much better chance of easing our user’s pain.

8. Conclusion

While this is a relatively long paper, there is so much more I could write. While this paper is no substitute for attending my Advanced Reactive Performance Management class [1], it should give you a solid grasp of the concepts and help you get started doing Oracle response time analysis.

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.
9. References


10. Acknowledgments

A special thanks to my email acquaintances, 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.

11. About the Author

Quoted a being "An Oracle performance philosopher who has a special place in history of Oracle performance management," Mr. Shallahamer brings his unique experiences to many as a keynote speaker, a sought after teacher, a researcher and publisher for ever improving Oracle performance management, and the founder of the grid computing company, BigBlueRiver. He is a recognized authority in the Oracle server technology community and is making waves in the grid community the result of founding a company which provides "Massive grid processing power—for the rest of us."

Mr. Shallahamer spent nine years at Oracle Corporation personally impacting literally hundreds of consultants, companies, database administrators, performance specialists, and capacity planners throughout the world. He left Oracle in 1998 to start OraPub, Inc. a company focusing on "Doing and helping others Do" both reactive and proactive Oracle performance management. He
continues to push performance management forward with his research, writing, consulting, highly valued teaching, and speaking engagements.

Combining his understanding of Oracle technology, the internet, and self organizing systems, Mr. Shallahamer founded BigBlueRiver in 2002 to help meet the needs of people throughout the world living in developing countries. People with limited technical and business skills can now start their own businesses which supply computing power into BigBlueRiver's computing grid. In a small way, this is making a difference in potentially thousands of people's lives.

Whether speaking at an Oracle, a grid computing, or a spiritual gathering, Mr. Shallahamer combines his experiences and his purpose toward communicating his unique insight into the technologies, the challenges, and the controversies of both Oracle and grid computing.