The Self-Managing Database: Automatic Health Monitoring and Alerting

Daniela Hansell, Oracle Corporation
Gaja Krishna Vaidyanatha, Oracle Corporation

Introduction

Enterprise databases continue to grow in size and number, resulting in increased systems management and administration complexity. Oracle Database 10*g* (henceforth denoted as Oracle 10*g* in this paper) introduces a host of self-managing capabilities to simplify administration, increase efficiency and lower the costs associated with systems management. Alert management is one area that has been completely revolutionized in Oracle 10*g*, empowering database administrators with proactive problem resolution, fault management, and automatic generation of "just in-time" alerts on various sub-systems.

This paper discusses a brand new method for Oracle database health monitoring called **server-generated alerts** along with its relevant architecture components. It also will discuss the sophisticated alert propagation framework of Oracle Enterprise Manager (EM) and how that is integrated into this new alerting method. The combination of EM and an efficient **server-generated alerts** system, delivers a complete system monitoring solution with a keen focus on the quality of service.

Why Do You Care?

Haven't you heard that the solution to all of life's problems is 42 [Adams,1]? So maybe you should set all of your alert thresholds to the magical number – 42 and you won't have to worry about anything. Better yet, we probably can do it for you automatically in a "self-managed" fashion. Maybe in the next release!

Jokes apart, a good portion of a database administrator's (DBA's) time is spent on monitoring database system's health, identifying bottlenecks and improving system performance. The detection of performance and other database-centric issues is fundamental to keeping systems up and running within reasonable service levels. A DBA needs to be informed of impending problems in a timely fashion, so that speedy diagnosis and problem resolution measures can be undertaken. No responsible DBA will want to live his or her life in reactive mode all the time. We all can tolerate only so many *bad hair days* in one year. **Server-generated Alerts** provide the necessary monitoring functionality with very miniscule levels of overhead.

Problems with Currently Available Monitoring Systems

There are many issues concerning database health monitoring solutions that are available in the market today. This section exposes some of them that which acted as driving factors behind the automatic health monitoring effort in Oracle 10*g*.

The Observer Affects The Observed (Macroscopic)

Werner Heisenberg (1901-1976) was a German physicist pioneered the formulation of some of the modern theories and principles of quantum physics in the beginning of the 20th century. **The Heisenberg Uncertainty Principle** was first presented in February 1927. *The Principle states that any attempted measurement of a particle or an object's position or velocity was noticeable on the object, at least at a subatomic scale.* This meant that it is impossible to determine both the exact position and the exact velocity of an object at the same time.

Okay, what does database health monitoring have to do with quantum physics and all this Geeky stuff? Actually, it has a lot in common. Read on!

The Uncertainty Principle – A Required Detour

Light (as we know it) is made up of energy packets called photons. To measure the position and velocity of any particle, one has to first shine light on an object or particle, and then measure the reflection of the light shone off the particle or object. On a macroscopic scale, the effect of photons on an object is insignificant.

Unfortunately, on subatomic scales, the photons that hit the subatomic particle will cause it to move significantly, so even though the position is measured accurately, the velocity of the particle may be altered. By determining the position of a subatomic particle, one can render any information previously collected on velocity completely useless. In other words, **the observer affects the observed**.

Database Health Monitoring And The Uncertainty Principle

The Uncertainty Principle is applicable to almost all walks of life. In the case of 3rd-party database health monitoring, the effects of the Uncertainty Principle are much more at the **macroscopic scale** contrasted with the subatomic scale observation done in the case of photons (or in Oracle 10*g*). When 3rd-party monitoring solutions affect their observed systems at significant levels, it causes us to believe that the Uncertainty Principle applies even at the macroscopic level, in the case of database health monitoring.

One corollary that can be derived from this is that, any effect the **observation process** imposes on the **observed** (database), should not be to the point that it affects the health of the observed. What we are talking about here as far as "affecting the observed" is not just resource overhead. It also includes resource contention that is generated. High levels of resource contention can potentially cripple observed systems, making them virtually unusable.

For example, a person who is very ill visits a healthcare professional, to determine the cause of his or her ill health. Given the state of the patient, the physician decides to *draw blood* to ensure that all vital statistics are within norm. *For reasons known only to the physician, blood is drawn every 15 minutes.* Here excessive resource consumption will eventually result in the death of the patient, as the human body's ability to create new blood cells will be outpaced by the quantity of blood drawn out. **The observer (physician) has affected the observed (patient)**. We differentiate here the difference between "overhead" and "runaway resource consumption".

Excessive or inefficient database health monitoring that cause unacceptable levels of resource consumption and contention on the target database and host. In doing so, it significantly **affect** the said targets. The targets are affected to levels that result in the death of an application or any underlying system component. *Server-generated alerts provide the solution to solve this problem, with an observance impact that can be measured only at the subatomic scale.*

Excessive Overhead

Historically, many monitoring solutions have used SQL to ping scores of database metrics and statistics, in an effort to monitor the health of databases. This is a prohibitively expensive monitoring method as this pings the database (**data pull**) for statistics in a persistent fashion. It is not uncommon to encounter performance-monitoring solutions that impose significant system overhead. We consider anything greater than 1% as **significant**.

In the recent past a new breed of monitoring solutions have become available, that utilize methods to directly attach to the Oracle System Global Area (SGA). The primary driving force (or selling point) in these solutions is that, in attaching to the Oracle SGA directly any overhead posed by the SQL layer can be eliminated. True, most of the time.

However, it should be kept in mind that direct-SGA attached solutions do not operate ***overhead free***. Albeit this method does not utilize the SQL layer and impose more SQL-based overhead on the system, direct-SGA attached monitoring solutions do consume significant amounts of CPU on monitored systems. This comes with the territory, as these monitors need to traverse through various C structures within the Oracle's shared memory areas and **sample** the relevant information at high rates. It is not uncommon to have sub-second sampling of values within Oracle's memory structures.

Complexity of Setup & Required Customization

Or, when was the last time you attempted to setup a VCR? In your effort to get rid of the annoying flashing "12:00", set the correct time, and record the next episode of the "Bachelor", you pretty much drove yourself to the brink of insanity. Sometimes even with the manuals! OK, maybe it was the right time for you to invest in a Tivo box. So we digress…

Most system monitoring solutions available in the market today require significant amounts of time and effort in setup. In the past we have known 3rd-party vendors selling monitoring systems that involved at least a week's worth of consulting time to setup the monitoring environment. Ouch!

The complexity of the setup for database health monitoring is usually characterized by the following:

- Too many scripts to support
- Many schema objects (tables, indexes, views, synonyms etc.)
- Deployment inflexibilities (too much manual configuration and task repetition)
- Lack of scalability (bad performance on mass deployment)

Lack Of Transition From Problem Detection To Diagnosis And Resolution

If you or someone you know suffer frequent skull-splitting headaches, it may be time to pay a visit to the doctor's office and determine the actual cause. Hopefully your physician will not draw blood every 15 minutes! While you may have consumed adequate amounts of pain medication to alleviate the symptoms, if the symptoms persist, the actual cause must be ascertained.

Similarly, it is not enough for performance monitors to just state that there are problems. They should not only detect conditions or problems, but also provide adequate information for accurate diagnosis and resolution of the identified problems. Bottom line, monitoring solutions should not just deal with symptoms, they need to facilitate diagnosis and curing of the underlying performance or configuration disease.

Benefits of Oracle 10*g*'s Automatic Health Monitoring: Server-Generated Alerts

The benefits of Oracle 10*g*'s **server-generated alerts** are many. This section outlines the salient factors that contribute to efficient and meaningful database health monitoring. Server-generated alerts are Oracle's next generation alerting technology that provides the user with relevant database health information.

The Observer Affects The Observed (Subatomic)

So what is the most significant benefit of **server-generated** alerts? More of the system resources are available to perform useful work that benefits your business and your enterprise. Server-generated alerts also generate zero contention on the available scarce resources on the target host and database. When compared with 3rd-party solutions, Oracle 10*g*'s server-side impact of "observing" is **subatomic**. The reason - *We are the object or the particle that is being observed.*

Extremely Low Overhead

The overhead of generating and delivering alerts including the cost of gathering all the required statistics is less than 0.1% of configured system resources. This contrasts the overhead posed by many 3rd-party monitoring solutions. The reason for such a low overhead is because the monitoring functionality is "built-in". There is no need for Oracle to run SQL or scan various memory structures and sample them at high rates, to gather database health information.

Server-generated alerts are "just-in-time". Plus, with the many years worth of system performance optimization expertise that has been incorporated into Oracle 10*g*'s automatic performance monitoring system, server-generated alerts and the 10*g* Advisors increase the probability of accurately detecting real problems, followed by relevant diagnosis and resolution.

Efficient Data Pushing

Unlike the traditional data pull method where data values for a given metric was requested (pinged) at frequent intervals, the **push method** provides the data value for a given metric **ONLY** when a problem is detected. This is possible because the monitoring functionality is embedded within the Oracle executable and is not an external SQL-based performance monitor. When problem conditions are detected, Oracle generates the required alert and pushes it into an advanced queue. Enterprise Manager automatically subscribes to this queue during the database discovery process and provides the necessary notification for that database.

For example if Oracle detects USER_WAIT_TIME_PCT (Wait Time Percentage for User Sessions) has exceeded a certain threshold, the relevant notifications are sent. Thus the need to constantly ping the database for the current values of hundreds of metrics and statistics, to determine whether or not there is a "performance problem", is completely eliminated. This results in saving scarce system resources for functions that benefit the business.

Minimal Configuration

As mentioned in previous sections, server-generated alerts is available on installation of Oracle 10*g* and does not require any additional configuration or setup. It is part of the Oracle Kernel and where relevant will have default threshold values already set. In the coming releases, Oracle will default many threshold values to meaningful and appropriate numbers (based on system workload). The Oracle Enterprise Manager 10*g* Database Control is installed out of the box and has alerts propagated and communicated on the Home Page. This way a DBA instantly benefits from the automated, low-overhead, out-of-the-box health monitoring and alerting of Oracle 10*g*. It does not get better than that!

Seamless Transition From Problem Detection To Diagnosis And Resolution

The self-managing database Oracle 10*g*, comes with both an alerting and advisor infrastructure that facilitate the transition from problem detection to timely diagnosis and resolution. The alerting mechanism is well integrated with the advisory framework and is able to generate context-sensitive diagnosis of problems, followed by relevant resolution methods, using advisors.

Automatic Database Health Monitoring In Oracle 10*g*

Oracle 10*g* raises the bar on database performance management by providing a built-in, low overhead and minimal configuration health monitoring functionality. Automatic Health Monitoring is part of the database install and it comes with a common infrastructure for all server components to deliver notifications and suggestions to external clients in both reactive and proactive modes.

The Intelligent Infrastructure

Oracle 10*g* is the first RDBMS to focus on manageability. Part of this effort was building an intelligent infrastructure to support revolutionary functionality enabling the zenith of self-monitoring, self-diagnostic and self-tuning databases. A new background process MMON and various other components such as the Automatic Workload Repository (AWR), Automatic Database Diagnostic Monitor (ADDM), Active Sessions History (ASH), form the building blocks of automatic monitoring.

MMON

Oracle 10*g*, introduces a dedicated manageability background process, MMON, intended for handling all the automatic management within the server. Using MMON, components in the database server can schedule monitoring actions to be performed periodically. Any component that detects a problem may either schedule a corrective action to be automatically executed by the server, or generate an alert message for the user to act upon.

Similarly, if a foreground process (also known as a server process) discovers some unusual condition, it can also invoke an urgent action, to be run by MMON. The action in turn generates an alert message to be sent to the user. In both cases, these alert messages are pushed to the user in a reliable and timely manner. The alert message contains the description of the problem and advice (where applicable) on how to fix it.

MMON also periodically flushes metrics (derived values of system-collected statistics) to a server built-in repository and maintains a history of their values. These metrics are not new to any DBA - Does *Buffer Cache Hit (%)* sound familiar? The metrics collected by the Oracle 10*g* server are a superset of what Enterprise Manager polled for in previous versions.

Automatic Workload Repository (AWR)

As self-managing components go through the cycle of problem detection, diagnosis and resolution, the accuracy of these actions heavily rely on the availability of comprehensive system performance statistics. AWR, a new server built-in facility that efficiently captures and maintains performance statistics, is an integral part of the management and monitoring infrastructure. AWR is a low-overhead data warehouse of the database that is automatically maintained. *A snapshot is any set of metrics, high-load SQL and hot objects collected at a given time and saved to the AWR.*

ASH

The Active Sessions History is the in-memory representation of active sessions along with their wait events and SQL information. It is maintained in a circular buffer for 30 minutes before it is flushed to disk, to AWR. ASH is Oracle's built-in direct-SGA attached performance data collector.

Advisors

Advisors are server centric modules that provide recommendations, within their realm, to improve resource utilization and performance for a particular database sub-component. Advisors provide the DBA with a wealth of context-sensitive information, related to the problems that are encountered. They complement the alert mechanism. The advisors may reference historical data in addition to current in-memory data in order to produce their recommendations. They are especially powerful because they provide vital tuning information that cannot be obtained any other way.

Performance Advisors such as the ADDM, SQL Tuning, SQL Access, Memory, Undo, and Segment are essential in identifying system bottlenecks and getting tuning advice for specific areas, on probable resolution paths.

Definition Of Terms

The database health-monitoring infrastructure feels right at home in this environment. Let us take a look at it in more detail and first understand the terminology.

Server-Generated Alerts

Server-Generated Alerts are messages that carry information about some event or state of an object notifying you of an impending problem, plus an optional suggestion for corrective action. The suggested action is a description of what the receiving user may perform to resolve the reported problem. What distinguishes them from traditional alerts is the fact that the database itself generates them in the most efficient and timely way possible by accessing the SGA directly.

Thresholds

Most alerts are associated with a calculated metric value. A boundary value of a metric is a threshold**.** Once this boundary is crossed, the system may be considered in undesirable state. Thresholds are further classified as *Warning* (less serious) and *Critical* (what you consider "really bad" for your business). If the actual value of the attribute/metric violates the threshold value a set number of times *(consecutive occurrences*) during a set period of time *(observation period),* an alert is issued. This ensures that we do not generate false alerts and intelligently handle occasional spikes. An AWR snapshot can be used to set appropriate threshold values for your system.

ARCHITECTURE

And now on to even more geeky stuff – how does this "too-good-to-be-true" solution work and why should you trust it? Alerts are by nature associated with "evil." So how can you love someone who always brings you bad news? Timely triggered alerts can be a DBA's friend. They anticipate problems before they happen. Not quite pre-cogs, but close (From the blockbuster movie - *Minority Report*).

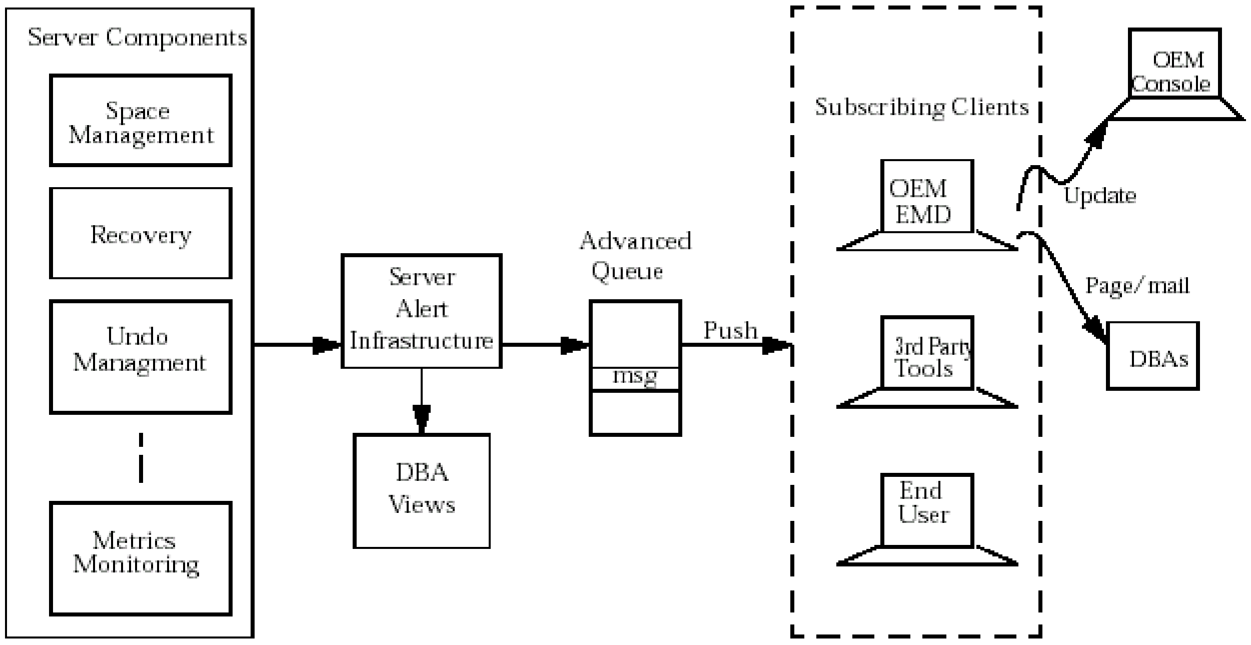


Figure 1

HOW DOES IT WORK?

In a nutshell, MMON wakes up every minute to compute metric values and checks for occurrences of events. If some of the metrics have internal or customer defined thresholds, MMON compares them with the actual values and, if exceeded, it generates alerts. If a special event occurred, MMON also produces an alert.

An Oracle database alert includes the identity of the object on which the alert was produced, the severity of the problem, a description of it, advice on a corrective action, and optionally the advisor name for more detailed advice. Objects can be any database entity, for example tablespaces.

An outstanding alert is queued into the predefined persistent advanced queue (ALERT_QUE) owned by SYS. When an alert occurs, MMON starts flushing to the Workload Repository the corresponding metric history for one hour before the alert has been raised and two minutes after the alert has been cleared. This fine-grained information is essential in producing recommendations and resolving the problem using the appropriate Advisors.

If for any reason an alert cannot be written to the Alert queue, a message containing information about that alert is written to the Database Alert log. Certain events such as lack of storage space or media corruption can cause Oracle not to write to the alert queue. In those cases, the database alert log is used as a secondary data persisting mechanism.

The consumers of the alert queue, such as Enterprise Manager (see *Figure 1* above) deliver the outstanding alerts. Depending on the their setup administrators can be visually notified on the managing Console, using e-mail, or a pager. By default, server alerts are always displayed on the Enterprise Manager Console. It is useful to note here that server-generated alerts, is the single source of truth with regards to database health information. Everything comes from one source and the infrastructure supports multiple subscribers consuming the same information, with no additional impact or overhead.

The severity level of an outstanding alert is periodically evaluated. When the current status of the problem changes, a new alert message is sent out containing the updated severity level. When the problem condition is cleared, the outstanding alert is moved to the alert history. The alert history is purged according to the AWR snapshot purging policy. By default, all alerts (both resolved and unresolved) are purged after seven days.

Type Of Alerts

There are two kinds of server generated alerts: *threshold-based* and *non–threshold based.*

Most database-generated alerts are threshold-based and are configured by setting threshold values on database metrics: a *Warning* and a *Critical* threshold. An alert is of the *Warning* type if the warning threshold has been exceeded, and of type *Critical* if the actual value has exceeded the critical one.

There are many metrics that are instance specific and can be customized by users. Some examples are – *Active Sessions Waiting for I/O, Active Sessions Using CPU, Active Sessions Waiting for Non-I/O Events*, etc. Some alerts are database wide alerts, such as alerts based on the *Tablespace Space Usage* metric.   This ensures that no duplicate alerts are issued in a multi-instance environment such as a Real Application Cluster (RAC).

Non–threshold-based alerts correspond to specific database events such as: *Snapshot Too Old* or *Resumable Session Suspended.* In this case, an alert simply indicates that the event has occurred.

Due to a wide variety of production systems, configurations and workloads, in most cases automatic setting of default threshold values does not make sense. By default, Oracle enables the *Tablespace Space Usage* alerts (Warning at 85% full, Critical at 97% full), *Snapshot Too Old*, *Recovery Area Low On Free Space* and the *Resumable Session Suspended.* Other alerts are enabled out of the box by the Enterprise Manager alert framework. As mentioned before, Oracle will in the coming releases, set default values for many thresholds, so that they are meaningful and relevant and based on actual system workloads.

Viewing Alert Metadata

As mentioned before, all alerts are published in the queue - ALERT_QUE. Thresholds-based alerts appear in the DBA_OUTSTANDING_ALERTS view, and when cleared they are moved to the DBA_ALERT_HISTORY table. Non-threshold-based alerts are automatically cleared after they are issued, and they move directly to the history table.

The in-memory values of system-level metrics can be accessed through the V$SYSMETRIC and V$SYSMETRIC_HISTORY views. By simply enabling the automatic snapshot collection to AWR, on-disk metric collection for all metrics is enabled. The on-disk values for the metrics are viewable through the DBA_HIST_* views.

Additional dictionary views that allow access to server alerts information:

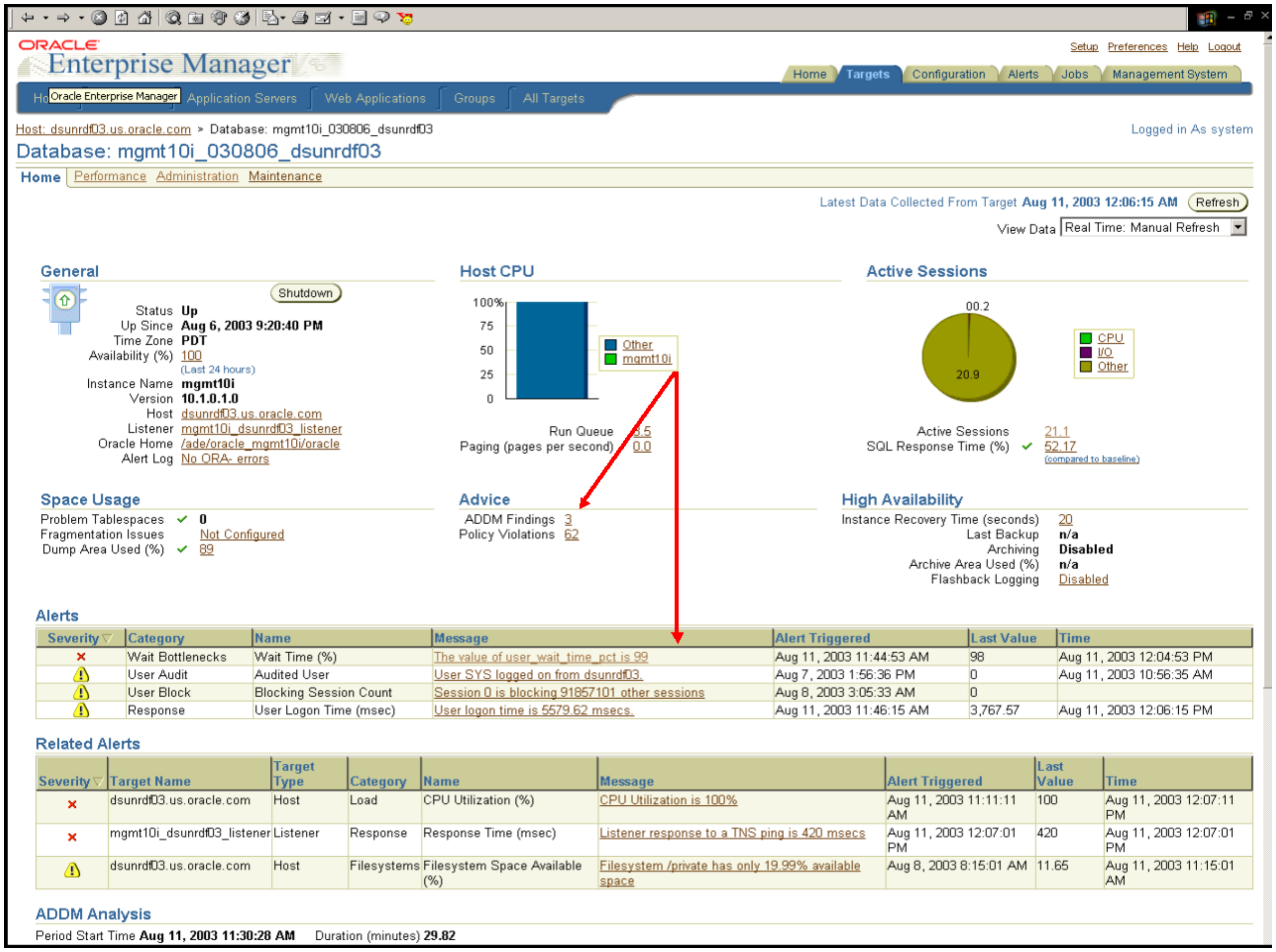
- DBA_THRESHOLDS, threshold settings defined for the instance.
- V$ALERT_TYPES, information on each alert type.

ORACLE ENTERPRISE MANAGER 10G DATABASE CONTROL - THE REALLY GOOD NEWS!

To avoid downtime and performance degradation, and provide full support for 27x7 applications, Enterprise Manager's alert infrastructure and graphical user interface enables proactive user notifications via paging, email or SNMP. Administrators can also define response actions for each alert condition, which can run automatically to correct a problem. In EM, you have the facility to create "Response Actions" that can be initiated on certain alert conditions.

The following use-case scenario illustrates how server-generated alerts and EM provide insight into an impending performance problem:

a. The database home page provides a high-level overview of the alerts generated for that database and also any performance advisor findings. There is a **Critical** alert for Wait Bottlenecks, where the percentage of wait is greater than 99%. There are also ADDM findings that provide the user context-sensitive performance tuning advice. Figure 2 illustrates that:



ORACLE Enterprise Manager
Setup Preferences Help Logout
Home Targets Configuration Alerts Jobs Management System
Oracle Enterprise Manager Application Servers Web Applications Groups All Targets
Shutdown
Refresh
Other
mgmt10i
CPU
I/O
Other
00.2
20.9
100%
75
50
25
0

Host: dsunrdf03.us.oracle.com » Database: mgmt10i_030806_dsunrdf03

Logged in As system

Database: mgmt10i_030806_dsunrdf03

Home Performance Administration Maintenance

Latest Data Collected From Target **Aug 11, 2003 12:06:15 AM**

View Data Real Time: Manual Refresh

General

Status	**Up**
Up Since	**Aug 6, 2003 9:20:40 PM**
Time Zone	**PDT**
Availability (%)	100 (Last 24 hours)
Instance Name	**mgmt10i**
Version	**10.1.0.1.0**
Host	dsunrdf03.us.oracle.com
Listener	mgmt10i_dsunrdf03_listener
Oracle Home	/ade/oracle_mgmt10i/oracle
Alert Log	No ORA- errors

Host CPU

Run Queue 3.5
Paging (pages per second) 0.0

Active Sessions

Active Sessions 21.1
SQL Response Time (%) ✓ 52.17 (compared to baseline)

Space Usage

Problem Tablespaces	✓	0
Fragmentation Issues		Not Configured
Dump Area Used (%)	✓	89

Advice

ADDM Findings 3
Policy Violations 62

High Availability

Instance Recovery Time (seconds)	20
Last Backup	**n/a**
Archiving	**Disabled**
Archive Area Used (%)	**n/a**
Flashback Logging	Disabled

Alerts

Severity	Category	Name	Message	Alert Triggered	Last Value	Time
✕	Wait Bottlenecks	Wait Time (%)	The value of user_wait_time_pct is 99	Aug 11, 2003 11:44:53 AM	98	Aug 11, 2003 12:04:53 PM
⚠	User Audit	Audited User	User SYS logged on from dsunrdf03.	Aug 7, 2003 1:56:36 PM	0	Aug 11, 2003 10:56:35 AM
⚠	User Block	Blocking Session Count	Session 0 is blocking 91857101 other sessions	Aug 8, 2003 3:05:33 AM	0	
⚠	Response	User Logon Time (msec)	User logon time is 5579.62 msecs.	Aug 11, 2003 11:46:15 AM	3,767.57	Aug 11, 2003 12:06:15 PM

Related Alerts

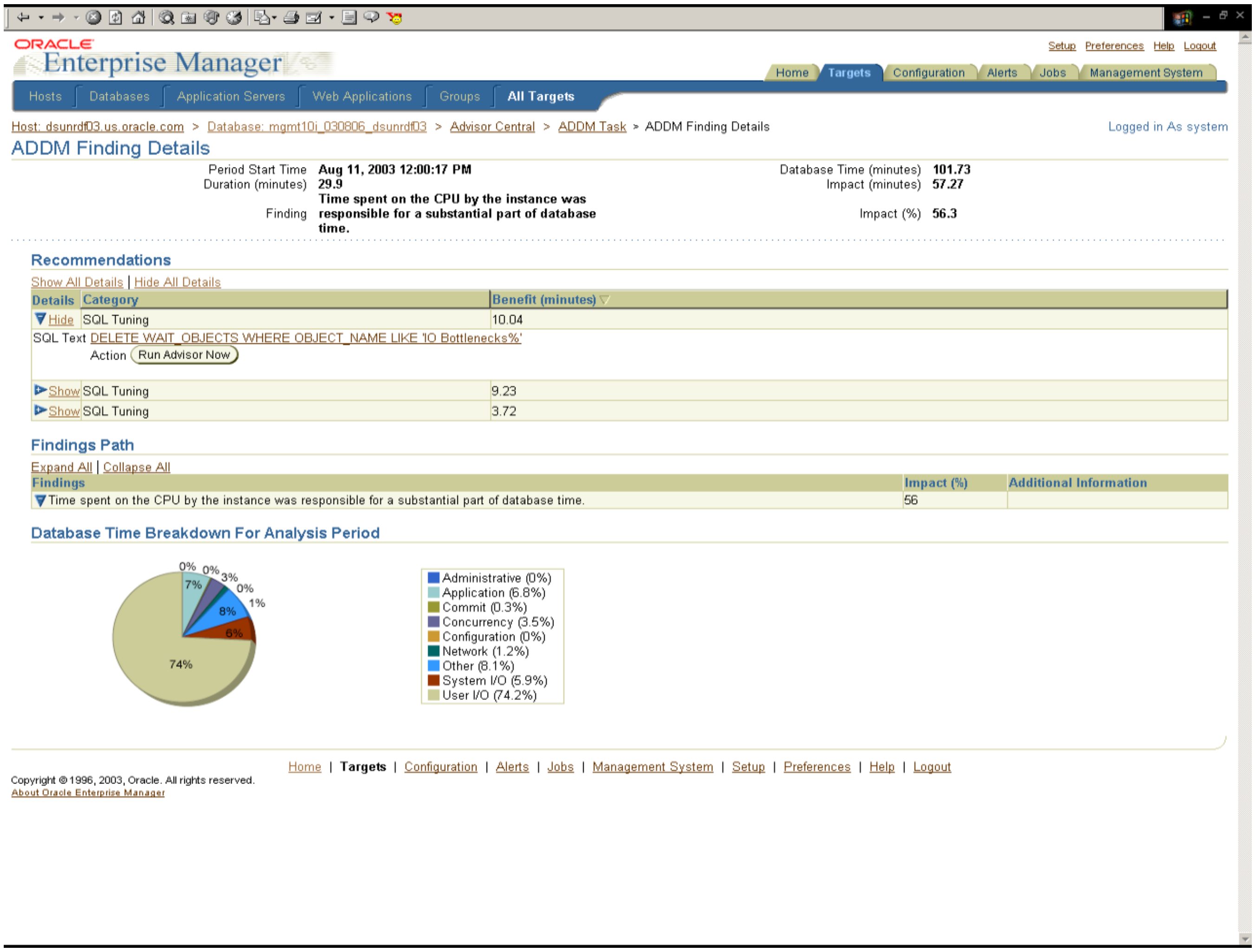
Severity	Target Name	Target Type	Category	Name	Message	Alert Triggered	Last Value	Time
✕	dsunrdf03.us.oracle.com	Host	Load	CPU Utilization (%)	CPU Utilization is 100%	Aug 11, 2003 11:11:11 AM	100	Aug 11, 2003 12:07:11 PM
✕	mgmt10i_dsunrdf03_listener	Listener	Response	Response Time (msec)	Listener response to a TNS ping is 420 msecs	Aug 11, 2003 12:07:01 PM	420	Aug 11, 2003 12:07:01 PM
⚠	dsunrdf03.us.oracle.com	Host	Filesystems	Filesystem Space Available (%)	Filesystem /private has only 19.99% available space	Aug 8, 2003 8:15:01 AM	11.65	Aug 11, 2003 11:15:01 AM

ADDM Analysis

Period Start Time **Aug 11, 2003 11:30:28 AM** Duration (minutes) **29.82**

Figure 2

b. Clicking on the "ADDM Findings", takes us to the ADDM Findings Details page. This page reveals that the offending SQL statement that was causing the problem is a DELETE. It is also very obvious (from the pie chart) that the User I/O Wait Class is consuming 74% of database time. These findings are from the latest ADDM analysis run, which is scheduled to run by default, every 30 minutes. Figure 3 provides more information:



ORACLE Enterprise Manager
Setup Preferences Help Logout
Home Targets Configuration Alerts Jobs Management System
Hosts Databases Application Servers Web Applications Groups All Targets

Host: dsunrdf03.us.oracle.com > Database: mgmt10i_030806_dsunrdf03 > Advisor Central > ADDM Task > ADDM Finding Details   Logged in As system

ADDM Finding Details

Period Start Time	**Aug 11, 2003 12:00:17 PM**	Database Time (minutes)	**101.73**
Duration (minutes)	**29.9**	Impact (minutes)	**57.27**
Finding	**Time spent on the CPU by the instance was responsible for a substantial part of database time.**	Impact (%)	**56.3**

Recommendations

Show All Details | Hide All Details

Details	Category	Benefit (minutes)
Hide	SQL Tuning	10.04
SQL Text	DELETE WAIT_OBJECTS WHERE OBJECT_NAME LIKE 'IO Bottlenecks%'	
	Action: Run Advisor Now	
Show	SQL Tuning	9.23
Show	SQL Tuning	3.72

Findings Path

Expand All | Collapse All

Findings	Impact (%)	Additional Information
Time spent on the CPU by the instance was responsible for a substantial part of database time.	56	

Database Time Breakdown For Analysis Period

- Administrative (0%)
- Application (6.8%)
- Commit (0.3%)
- Concurrency (3.5%)
- Configuration (0%)
- Network (1.2%)
- Other (8.1%)
- System I/O (5.9%)
- User I/O (74.2%)

Home | Targets | Configuration | Alerts | Jobs | Management System | Setup | Preferences | Help | Logout

Copyright © 1996, 2003, Oracle. All rights reserved.
About Oracle Enterprise Manager

Figure 3

c. To determine the performance issue that is currently manifesting, the Performance Page (Figure 4) is accessed. This reveals the various "Wait Classes" that are accumulating waits for the active sessions on the database. It is observed here that the wait class **User I/O** is consuming the largest amount of waits. The host is also pegged on the CPU resource.

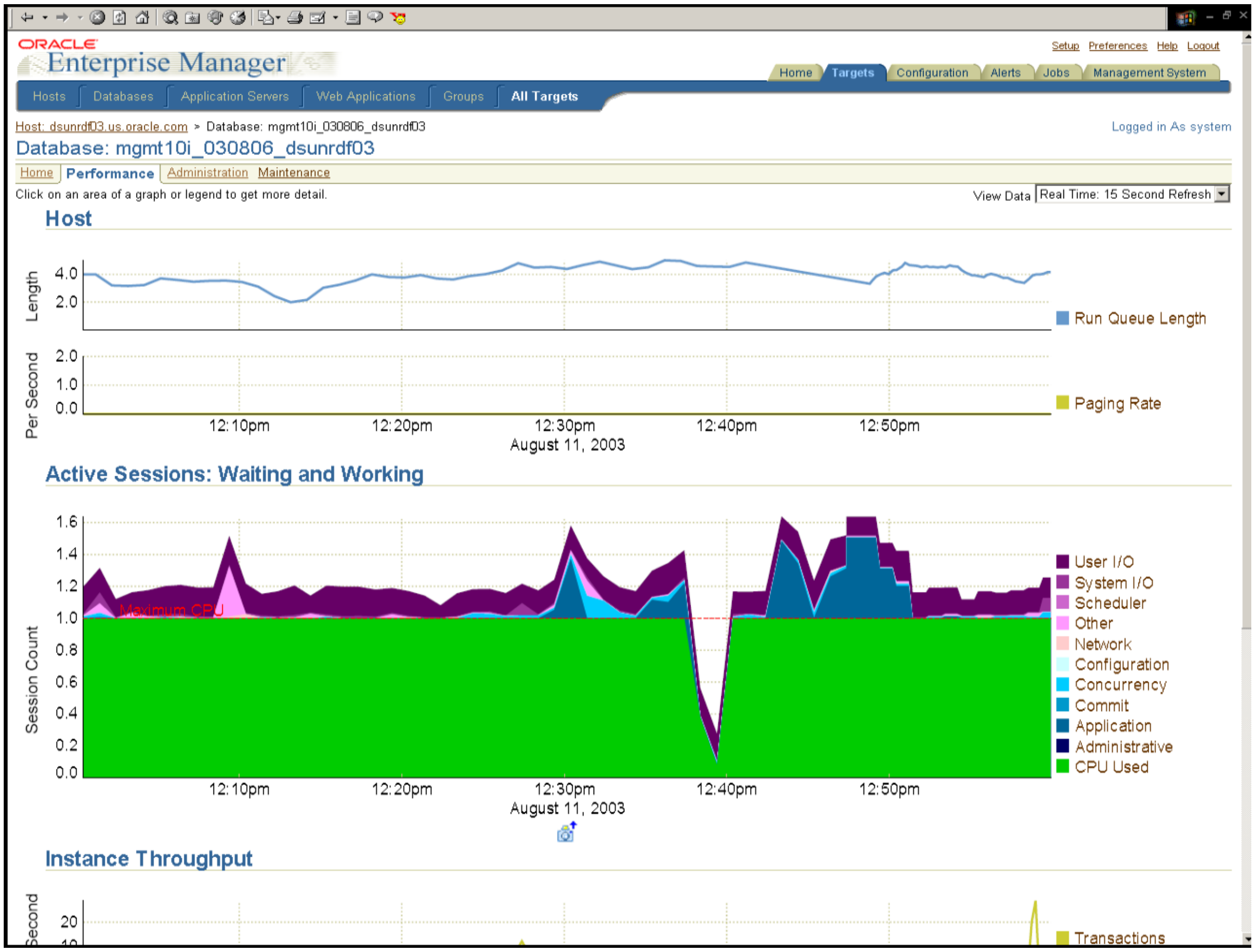
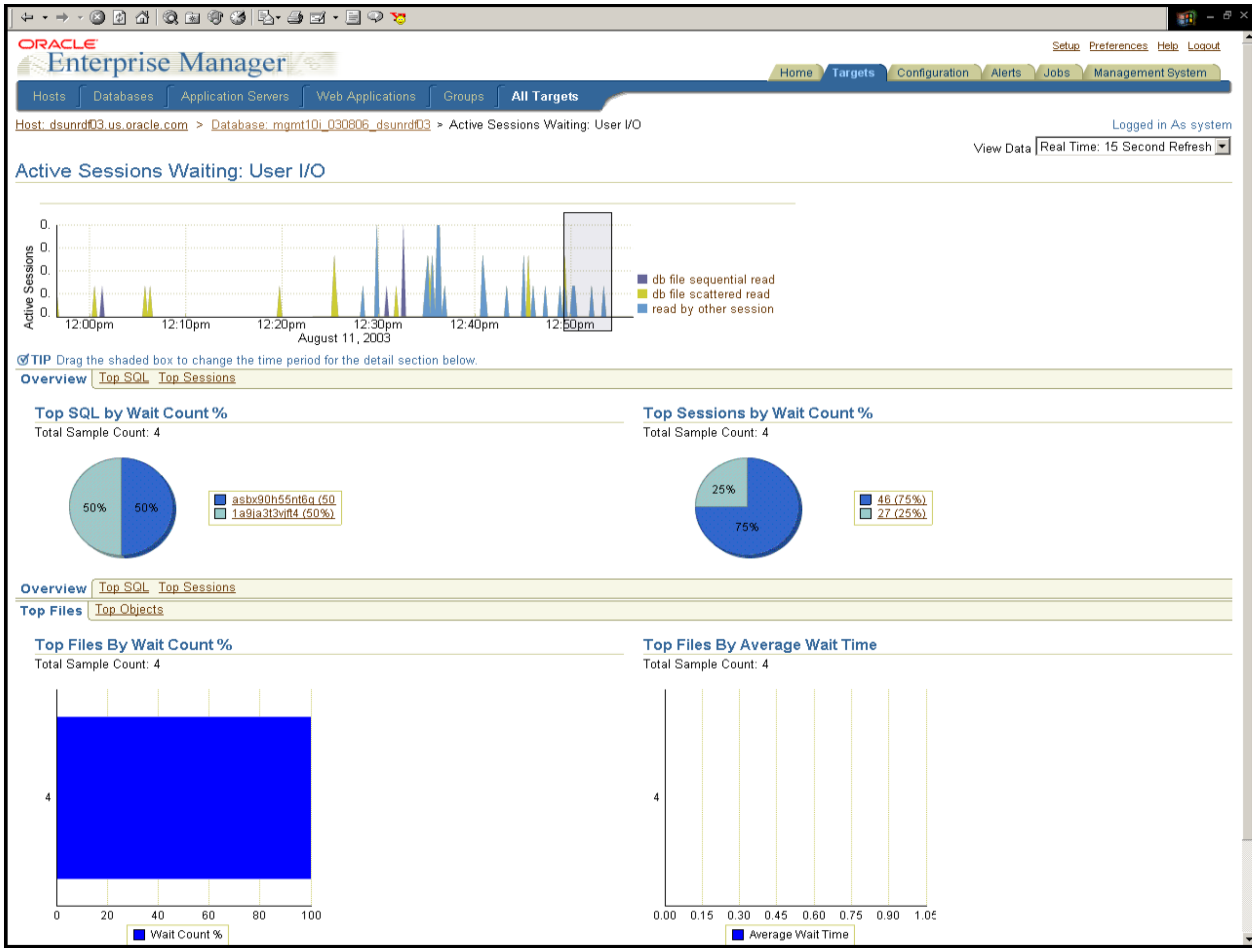

ORACLE
Enterprise Manager
Setup Preferences Help Logout
Home Targets Configuration Alerts Jobs Management System
Hosts Databases Application Servers Web Applications Groups All Targets
Host: dsunrdf03.us.oracle.com > Database: mgmt10i_030806_dsunrdf03
Logged in As system
Database: mgmt10i_030806_dsunrdf03
Home Performance Administration Maintenance
Click on an area of a graph or legend to get more detail.
View Data Real Time: 15 Second Refresh
Host
Length
Per Second
Run Queue Length
Paging Rate
12:10pm 12:20pm 12:30pm 12:40pm 12:50pm
August 11, 2003
Active Sessions: Waiting and Working
Maximum CPU
Session Count
User I/O
System I/O
Scheduler
Other
Network
Configuration
Concurrency
Commit
Application
Administrative
CPU Used
Instance Throughput
Transactions

Figure 4

d. Clicking on the User I/O legend tag or the "purple" band of color (which is stacked on the top of the chart) navigates us to the drilldown for the User I/O. This enables us to find out the Top Sessions and the Top SQL statements that are currently plaguing the system. Figure 5 does the rest:


ORACLE
Enterprise Manager
Setup Preferences Help Logout
Home Targets Configuration Alerts Jobs Management System
Hosts Databases Application Servers Web Applications Groups All Targets
Host: dsunrdf03.us.oracle.com > Database: mgmt10i_030806_dsunrdf03 > Active Sessions Waiting: User I/O
Logged in As system
View Data Real Time: 15 Second Refresh
Active Sessions
12:00pm
12:10pm
12:20pm
12:30pm
12:40pm
12:50pm
August 11, 2003
Overview
Top SQL
Top Sessions
Top Files
Top Objects
Wait Count %
Average Wait Time

Active Sessions Waiting: User I/O

db file sequential read
db file scattered read
read by other session

TIP Drag the shaded box to change the time period for the detail section below.

Top SQL by Wait Count %
Total Sample Count: 4

- asbx90h55nt6q (50
- 1a9ja3t3vjft4 (50%)

Top Sessions by Wait Count %
Total Sample Count: 4

- 46 (75%)
- 27 (25%)

Top Files By Wait Count %
Total Sample Count: 4

Top Files By Average Wait Time
Total Sample Count: 4

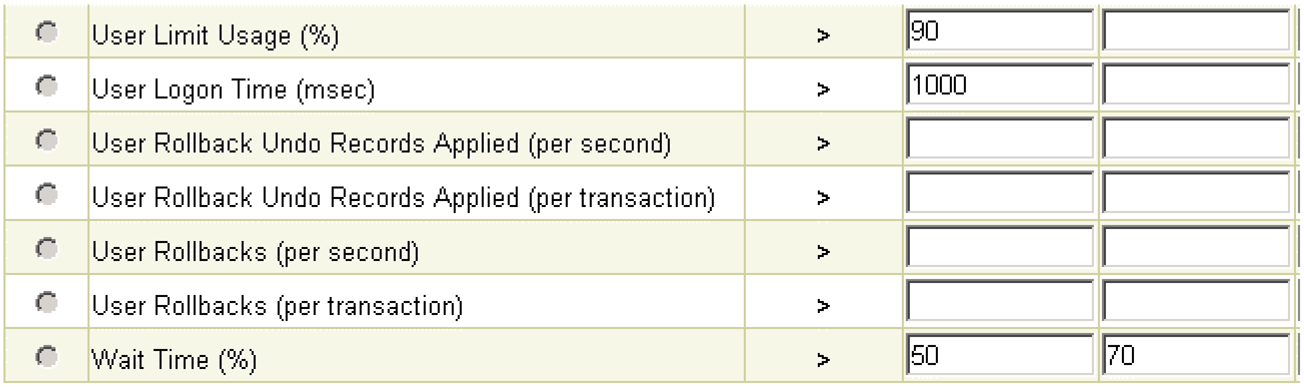
Figure 5

e. As a resolution step to this problem, the user is led to run the SQL Tuning Advisor. Refer to Figure 3, under "Recommendations", there is a button for running SQL Tuning Advisor for the DELETE statement. If you are curious, how the problem was solved, SQL Tuning Advisor recommended that we create an index on the WAIT_OBJECTS.OBJECT_NAME column.

f. Mission Accomplished! The server-generated alerts and the EM Database Console have teamed up to perform problem detection, accurate diagnosis and timely resolution of a performance problem.

SETTING THRESHOLDS

Enterprise Manager's graphical interface facilitates easy configuration and management of threshold values where required. Figure 6 shows how:



○	User Limit Usage (%)	>	90	
○	User Logon Time (msec)	>	1000	
○	User Rollback Undo Records Applied (per second)	>		
○	User Rollback Undo Records Applied (per transaction)	>		
○	User Rollbacks (per second)	>		
○	User Rollbacks (per transaction)	>		
○	Wait Time (%)	>	50	70

Figure 6

SERVER-GENERATED ALERTS – PL/SQL APIS

If you are one of those developer types or just love command-line interfaces (CLIs) and would like to know how to do everything that EM has effortlessly done for you, this section will be of great interest. Here we provide you with pertinent information to use server-generated alerts along with a recipe that contains sample (working) code that you can re-use.

MANAGING THRESHOLDS

Thresholds are commonly set after you have observed your working systems over a period of time and decided on appropriate values for each metric. The Oracle 10*g* alert infrastructure provides PL/SQL interfaces to manipulate their values. The DBMS_SERVER_ALERT PL/SQL package can used to retrieve currently set threshold values and set new thresholds on available metrics. This package include the following procedures:

- GET_THRESHOLD to read settings for a given metric
- SET_THRESHOLD to define new threshold settings for a given metric

For example, to set the Warning threshold at 60% and Critical threshold at 80% for the tablespace KITCHEN you can do the following:

```
SQL> exec DBMS_SERVER_ALERT.SET_THRESHOLD(9000,DBMS_SERVER_ALERT.OPERATOR_GE,'60',
DBMS_SERVER_ALERT.OPERATOR_GE,'80',1,1,NULL,DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE
,'KITCHEN');
```

A Warning alert will be issued when 60% of the tablespace in the KITCHEN is used. Similarly, a Critical alert will be issued when 80% of the tablespace in the KITCHEN is used. Get it!

The other input parameters map to the following:

- External metric identifier, in this case, 9000, can be retrieved from METRIC_ID in V$METRICNAME.

- Operator comparing the actual value for the Warning Threshold value (for example, *Greater Than or Equal To*). The constant that defines "Greater Than Or Equal To" is DBMS_SERVER_ALERT.OPERATOR_GE. Similarly for "Less Than", the constant is DBMS_SERVER_ALERT.OPERATOR_LT.
- Warning Threshold Value, in this case, it is set to 60.
- Operator comparing the actual value for the Critical Threshold value.
- Critical Threshold Value, in this case, it is set to 80.
- Observation period (in minutes), which defines for how long the actual behavior of the system, must deviate from the threshold value before the alert is issued. This provision ensures that alerts are not issued for short–term spikes. In this case the observation period is 1.
- Consecutive occurrences, defines how many times the metric value should violate the threshold values before the alert is issued. In this case the consecutive occurrences is set to 1.
- Name of the instance for which the threshold is set. Use the value NULL, for a single instance database.
- Object Type, type of the object on which the threshold is set, in this case, it is a tablespace defined by the constant DBMS_ALERT.OBJECT_TYPE_TABLESPACE.
- Name of the object, in this case, the tablespace name is KITCHEN.

CONSUMING ALERTS

Once alerts are issued and published in the Alerts queue, they are available for consumption by any subscriber with appropriate privileges. To retrieve the alerts and channel them through a notification mechanism, here is a typical sequence of steps. The recipe outlined below assumes that you are logged in as SYS and the SYSTEM database user is consuming the alerts.

1. ***Subscribe to ALERT_QUE***: Users with execution privilege of DBMS_AQADM can subscribe to the ALERT_QUE using DBMS_AQADM.ADD_SUBSCRIBER procedure.

```
/* The second argument of the following procedure is the subscribe, i.e., the agent
on whose behalf the subscription has been defined. The AQ$_AGENT procedure is used
to subscribe to the queue and the name assigned to the subscribing agent is
ALERT_USR1. */
SQL> exec DBMS_AQADM.ADD_SUBSCRIBER(queue_name=>'ALERT_QUE',
AQ$_AGENT('ALERT_USR1','',0));
PL/SQL procedure successfully completed.

/* The "agent" in this context is the subscribing user */
SQL> exec DBMS_AQADM.CREATE_AQ_AGENT(agent_name=>'ALERT_USR1');
PL/SQL procedure successfully completed.
```

2. ***Associate the database user with the subscribing agent***: Since the ALERT_QUE is a secure queue only the user associated with the subscribing agent can access any queued messages. This association can be done using the DBMS_AQADM.ENABLE_DB_ACCESS procedure and actual privilege can be assigned using the DBMS_ADQDM.GRANT_QUEUE_PRIVILEGE procedure.

```
SQL> exec
DBMS_AQADM.ENABLE_DB_ACCESS(agent_name=>'ALERT_USR1',db_username=>'SYSTEM');
PL/SQL procedure successfully completed.

SQL> exec
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(privilege=>'DEQUEUE',queue_name=>'ALERT_QUE',\
grantee=>'SYSTEM',grant_option=>FALSE);
PL/SQL procedure successfully completed.
```

3. ***Register for notifications***: Optionally, you may register to receive an asynchronous notification when an alert is enqueued to ALERT_QUE. The notification can be in the form of e-mail, HTTP post or PL/SQL procedure. The registration for notification can be done using DBMS_AQ.REGISTER procedure. Since ALERT_QUE is a persistent queue, the notification only includes the AQ metadata of the enqueue message.

```
SQL> DECLARE
       reginfo     aq$_reg_info;
       reginfolist aq$_reg_info_list;
     BEGIN
       reginfo := AQ$_REG_INFO('ALERT_QUE:ALERT_USR1',
       DBMS_AQ.NAMESPACE_AQ, 'mailto://janedoe@yourcompany.com',NULL);
       -- Create the registration info list
       reginfolist := AQ$_REG_INFO_LIST(reginfo);
       -- Register the registration info list
       DBMS_AQ.REGISTER(reginfolist, 1);
     END;
     /
PL/SQL procedure successfully completed.
```

4. ***Configure E-mail and HTTP proxy***: If you get enqueue notifications through e-mail or HTTP posts, you need to configure the database server through DBMS_AQELM with information about the mail server or proxy server, such as the mail host name, mail port number, proxy server name, etc.

```
SQL> BEGIN
       DBMS_AQELM.SET_MAILHOST('yourmailhost.com');
       DBMS_AQELM.SET_MAILPORT(25);
       DBMS_AQELM.SET_SENDFROM('janedoe@yourcompany.com');
       COMMIT;
     END;
PL/SQL procedure successfully completed.
```

5. ***Dequeue an alert***: Subscribers need to dequeue a message to read the contents of the message. You need *dequeue* privileges on the ALERT_QUE to dequeue an alert. A message can be dequeued using DBMS_AQ.DEQUEUE procedure or the OCIAQDeq call. Alert messages will remain in ALERT_QUE per each subscriber until the subscriber dequeues the message, or it expires.

```
SQL> DECLARE
       dequeue_options     dbms_aq.dequeue_options_t;
       message_properties  dbms_aq.message_properties_t;
       message             ALERT_TYPE;
       message_handle      RAW(16);
BEGIN
  dequeue_options.consumer_name := 'ALERT_USR1';
  /* Never wait */
  dequeue_options.wait := DBMS_AQ.NO_WAIT;
  /* Always reset the position to the begining of the AQ */
  dequeue_options.navigation := DBMS_AQ.FIRST_MESSAGE;
  /* Remove the message when done */
  dequeue_options.dequeue_mode := DBMS_AQ.REMOVE;
  /* set some filter condition
  dequeue_options.deq_condition := 'tab.user_data.sequence_id > 6'; */
  DBMS_AQ.DEQUEUE(
          queue_name          =>     'ALERT_QUE',
          dequeue_options     =>     dequeue_options,
          message_properties  =>     message_properties,
          payload             =>     message,
          msgid               =>     message_handle);

  /* The alert message dequeued output presented below is a subset of the available
  fields in the message queue */
  DBMS_OUTPUT.PUT_LINE('Alert message dequeued:');
  DBMS_OUTPUT.PUT_LINE('  Timestamp:         ' || message.timestamp_originating);
  DBMS_OUTPUT.PUT_LINE('  Organization Id:   ' || message.organization_id);
  DBMS_OUTPUT.PUT_LINE('  Component Id:      ' || message.component_id);
  DBMS_OUTPUT.PUT_LINE('  Message Type:      ' || message.message_type);
  DBMS_OUTPUT.PUT_LINE('  Message Group:     ' || message.message_group);
  DBMS_OUTPUT.PUT_LINE('  Message Level:     ' || message.message_level);
  DBMS_OUTPUT.PUT_LINE('  Host Id:           ' || message.host_id);
  DBMS_OUTPUT.PUT_LINE('  Host Network Addr: ' || message.host_nw_addr);
  DBMS_OUTPUT.PUT_LINE('  Reason:            ' ||

  DBMS_SERVER_ALERT.EXPAND_MESSAGE(userenv('LANGUAGE'),message.message_id,
  message.reason_argument_1,message.reason_argument_2,message.reason_argument_3,
  message.reason_argument_4,message.reason_argument_5));

  DBMS_OUTPUT.PUT_LINE('  Sequence Id:       ' || message.sequence_id);
  DBMS_OUTPUT.PUT_LINE('  Reason Id:         ' || message.reason_id);
  DBMS_OUTPUT.PUT_LINE('  Object Name:       ' || message.object_name);
  DBMS_OUTPUT.PUT_LINE('  Object Type:       ' || message.object_type);
  DBMS_OUTPUT.PUT_LINE('  Instance Name:     ' || message.instance_name);
  DBMS_OUTPUT.PUT_LINE('  Suggested action:  ' ||
  DBMS_SERVER_ALERT.EXPAND_MESSAGE(userenv('LANGUAGE'),
  message.suggested_action_msg_id,
  message.action_argument_1,message.action_argument_2,
  message.action_argument_3,message.action_argument_4, message.action_argument_5));
  DBMS_OUTPUT.PUT_LINE('  Advisor Name:      ' || message.advisor_name);
  DBMS_OUTPUT.PUT_LINE('  Scope:             ' || message.scope);
END;
/
PL/SQL procedure successfully completed.

/* The following output is formatted */

Alert message dequeued:
Timestamp:         08-AUG-03 13.43.15.392725 PM -07:00
Organization Id:   yourcompany.com
Message Type:      Notification
Message Group:     Configuration
Message Level:     32
Host Id:           acme
Host Network Addr: 555.555.555.555
Reason:            Threshold is updated on metrics "Tablespace Space Usage"
Sequence Id:       41
Reason Id:         137
Object Name:       KITCHEN
Object Type:       TABLESPACE
Instance Name:     inst1
Suggested action:  Check DBA_THRESHOLDS view to verify the result
Advisor Name:      Default Advisor
Scope:             Database
PL/SQL procedure successfully completed.
```

Sound good! Great! If all of this PL/SQL was a bit too much for you to handle, you always have EM to fall back on, which implements all of this behind the scenes and presents it to you in a browser-based interface.

CONCLUSION

Database Health Monitoring is a required component of systems performance management. Enterprises require early problem detection, followed by proper diagnosis and resolution. Over-alerting has historically plagued too many systems with high overhead and false alarms. **Server-generated alerts** truly depict **Heisenberg's Uncertainty Principle**, as the observation impact on the observed is only at the sub-atomic level. This is due to the fact that the observer is within the observed. With 3rd-party monitoring solutions, the Uncertainty Principle's effect is at the macroscopic level, as the observer is external to the observed.

Oracle 10*g*'s built-in server-generated alerts and Enterprise Manager's propagation framework along with its browser-based interface provide the foundation to manage systems performance problems and database maintenance tasks. This is done at a much lower cost (both in terms of money spent and system resources utilized) compared to traditional monitoring systems. In the end, the various self-managing initiatives provided by Oracle 10*g*, assist in enabling automation of Oracle systems reducing manual intervention, lowering costs and providing better quality of service.

CREDITS

We'd like to convey our special thanks to our colleagues at Oracle who tolerated us chasing them around at the last minute for reviewing this paper and providing relevant and constructive feedback. Here they are in alphabetical order:

John Beresniewicz

Jane Chen

Mark Ramacher

Leng Tan

Alex Tsukerman

Graham Wood

REFERENCES

1. Adams, D. *The Ultimate Hitchhiker's Guide to the Galaxy*. Del Rey, 2002. http://www.randomhouse.com