Database Performance Analysis Techniques Using Metric Extensions and SPA

Kurt Engeleiter
Oracle Corporation
Redwood Shores, CA, USA

Keywords:

ADDM, SQL Tuning Advisor, SQL Performance Analyzer, Metric Extensions

Introduction

Maintaining good database performance can be challenging. Oracle provides a number of tools to help you monitor, identify and fix database performance issues. This paper and presentation discuss some techniques of using ADDM, SQL Tuning Advisor, SQL Performance Analyzer and Metric Extensions to monitor and maintain your database performance.

ADDM

Automatic Database Diagnostic Monitor

When problems occur with a database, it is important to perform accurate and timely diagnosis of the problem before making any changes. Often a database administrator (DBA) simply looks at the symptoms and immediately starts changing the system to fix those symptoms. However, an accurate diagnosis of the actual problem in the initial stage significantly increases the probability of success in resolving the problem.

With Oracle Database, the statistical data needed for accurate diagnosis of a problem is stored in Automatic Workload Repository (AWR). Automatic Database Diagnostic Monitor (ADDM):

- Analyzes AWR data on a regular basis
- Diagnoses the root causes of performance problems
- Provides recommendations for correcting any problems
- Identifies non-problem areas of the system
-

Because AWR is a repository of historical performance data, ADDM can analyze performance issues after the event, often saving time and resources in reproducing a problem.
In most cases, ADDM output should be the first place that a DBA looks when notified of a performance problem.

ADDM provides the following benefits:

- Automatic performance diagnostic report every hour by default
- Problem diagnosis based on decades of tuning expertise
- Time-based quantification of problem impacts and recommendation benefits

- Identification of root cause, not symptoms
- Recommendations for treating the root causes of problems
- Identification of non-problem areas of the system
- Minimal overhead to the system during the diagnostic process

Tuning is an iterative process, and fixing one problem can cause the bottleneck to shift to another part of the system. Even with the benefit of ADDM analysis, it can take multiple tuning cycles to reach acceptable system performance. ADDM benefits apply beyond production systems; on development and test systems, ADDM can provide an early warning of performance issues.

ADDM Analysis

An ADDM analysis can be performed on a pair of AWR snapshots and a set of instances from the same database. The pair of AWR snapshots define the time period for analysis, and the set of instances define the target for analysis.

If you are using Oracle Real Application Clusters (Oracle RAC), then ADDM has three analysis modes:

- Database: In Database mode, ADDM analyzes all instances of the database.
- Instance: In Instance mode, ADDM analyzes a particular instance of the database.
- Partial: In Partial mode, ADDM analyzes a subset of all database instances.

If you are not using Oracle RAC, then ADDM can only function in Instance mode because only one instance of the database exists.

An ADDM analysis is performed each time an AWR snapshot is taken and the results are saved in the database. The time period analyzed by ADDM is defined by the last two snapshots (the last hour by default). ADDM will always analyze the specified instance in Instance mode. For non-Oracle RAC or single instance environments, the analysis performed in the Instance mode is the same as a database-wide analysis. If you are using Oracle RAC, then ADDM also analyzes the entire database in Database mode. After an ADDM completes its analysis, you can view the results using Cloud Control, or by viewing a report in a SQL*Plus session.

ADDM analysis is performed top down, first identifying symptoms, and then refining them to reach the root causes of performance problems. The goal of the analysis is to reduce a single throughput metric called DB time. DB time is the fundamental measure of database performance, and is the cumulative time spent by the database in processing user requests. It includes wait time and CPU time of all non-idle user sessions. DB time is displayed in the V$SESS_TIME_MODEL and V$SYS_TIME_MODEL views.

By reducing DB time, the database is able to support more user requests using the same resources, which increases throughput. The problems reported by ADDM are sorted by the amount of DB time they are responsible for. System areas that are not responsible for a significant portion of DB time are reported as non-problem areas.

The types of problems that ADDM considers include the following:

- CPU bottlenecks - Is the system CPU bound by Oracle Database or some other application?

- Undersized Memory Structures - Are the Oracle Database memory structures, such as the SGA, PGA, and buffer cache, adequately sized?
- I/O capacity issues - Is the I/O subsystem performing as expected?
- High-load SQL statements - Are there any SQL statements which are consuming excessive system resources?
- High-load PL/SQL execution and compilation, and high-load Java usage
- Oracle RAC specific issues - What are the global cache hot blocks and objects; are there any interconnect latency issues?
- Sub-optimal use of Oracle Database by the application - Are there problems with poor connection management, excessive parsing, or application level lock contention?
- Database configuration issues - Is there evidence of incorrect sizing of log files, archiving issues, excessive checkpoints, or sub-optimal parameter settings?
- Concurrency issues - Are there buffer busy problems?
- Hot objects and top SQL for various problem areas

ADDM also documents the non-problem areas of the system. For example, wait event classes that are not significantly impacting the performance of the system are identified and removed from the tuning consideration at an early stage, saving time and effort that would be spent on items that do not impact overall system performance.

Using ADDM with Oracle Real Application Clusters

If you are using Oracle RAC, then run ADDM in Database analysis mode to analyze the throughput performance of all instances of the database. In Database mode, ADDM considers DB time as the sum of the database time for all database instances. Using the Database analysis mode enables you to view all findings that are significant to the entire database in a single report, instead of reviewing a separate report for each instance.

The Database mode report includes findings about database resources (such as I/O and interconnect). The report also aggregates findings from the various instances if they are significant to the entire database. For example, if the CPU load on a single instance is high enough to affect the entire database, then the finding appears in the Database mode analysis, which points to the particular instance responsible for the problem.

SQL Tuning Advisor

SQL Tuning Advisor can be run manually or also can run automatically during system maintenance windows as a maintenance task. During each automatic run, the advisor selects high-load SQL queries in the database and generates recommendations for tuning these queries.

SQL Tuning Advisor recommendations fall into the following categories:

- Statistics analysis
- SQL profiling
- Access path analysis
- SQL structure analysis

A SQL profile contains additional statistics specific to a SQL statement and enables the optimizer to generate a better execution plan. Essentially, a SQL profile is a method for analyzing a query. Both

access path and SQL structure analysis are useful for tuning an application under development or a homegrown production application.

A principal benefit of SQL Tuning Advisor is that solutions come from the optimizer rather than external tools. Thus, tuning is performed by the database component that is responsible for the execution plans and SQL performance. The tuning process can consider past execution statistics of a SQL statement and customizes the optimizer settings for this statement.

SQL Performance Analyzer

Changes that affect SQL execution plans can severely impact application performance and availability. As a result, DBAs spend enormous amounts of time identifying and fixing SQL statements that have regressed due to the system changes. SQL Performance Analyzer (SPA) can predict and prevent SQL execution performance problems caused by environment changes.

SQL Performance Analyzer provides a granular view of the impact of environment changes on SQL execution plans and statistics by running the SQL statements serially before and after the changes. The SQL Performance Analyzer report that is generated outlines the net benefit on the workload due to the system change as well as the set of regressed SQL statements. For regressed SQL statements, appropriate execution plan details along with recommendations to tune them are provided.

SQL Performance Analyzer is well integrated with existing SQL Tuning Set (STS), SQL Tuning Advisor and SQL Plan Management functionality. SQL Performance Analyzer completely automates and simplifies the manual and time-consuming process of assessing the impact of change on extremely large SQL workloads (thousands of SQL statements). DBAs can use SQL Tuning Advisor to fix the regressed SQL statements in test environments and generate new plans. These plans are then seeded to SQL Plan Management baselines and exported back into production. Thus, using SQL Performance Analyzer, businesses can validate with a high degree of confidence that a system change to a production environment in fact results in net positive improvement at a significantly lower cost.

Examples of common system changes for which you can use the SQL Performance Analyzer include:

- Database upgrade, patches, initialization parameter changes
- Configuration changes to the operating system, hardware, or database
- Schema changes such as adding new indexes, partitioning or materialized views
- Gathering optimizer statistics.
- SQL tuning actions, for example, creating SQL profiles
- Database consolidation testing using Oracle Multitenant Databases or schema consolidation methods

SQL Performance Analyzer functionality is available through DBMS_SQLPA API and through Oracle Enterprise Manager interface. There are various custom workflows available through Enterprise Manager and the Guided Workflow encompasses all of them and provides complete control and flexibility at every step of the process. Using the SQL Performance Analyzer Guided Workflow involves the following 5 main steps:

1. Capture the SQL workload that you want to analyze with SPA. The Oracle database offers ways to capture SQL workload from several sources, such as cursor cache and Automatic Workload Repository, into a SQL Tuning Set (STS). This would typically be done on a

production system and the STS would then be transported to the test system where SPA analysis will take place.

2. Measure the performance of the workload before a change by executing SPA on the STS. Very short running queries are executed multiple times and their statistics are averaged to eliminate variations due to buffer cache state and other noise factors

3. Make the change, such as database upgrade or optimizer statistics refresh.

4. Measure performance of the workload after the change by executing SPA on the STS again, as in step 2.

5. Compare performance of the two executions of the SQL tuning set to identify the SQL statements that have regressed, improved, or were unchanged.

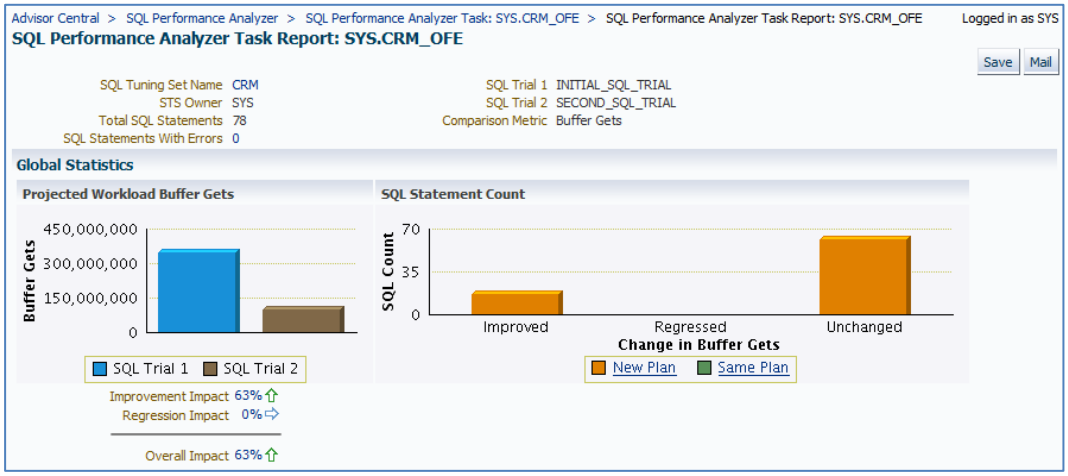

Illustration. 1: SPA Comparison Report

The SPA comparison report in illustration 1 shows significant performance improvement of overall SQL workload after the proposed system change but with a few execution plan regressions.
SQL Performance Analyzer takes into account the number of executions of a SQL statement when measuring its impact. A SQL statement that completes in seconds but is frequently executed may have a higher impact on the system than a long running statement executed only once. SPA takes these factors into account when predicting overall performance improvements and regressions. If any regressions are encountered, SPA allows the user to fix them using SQL Tuning Advisor or with SQL Plan Baselines, a plan stability feature introduced in Oracle Database 11g.

SPA supports numerous other features that help assess system changes, these are briefly described below:

1. SPA helps estimate the IO reduction that can be accomplished by migrating to Exadata server but without actually requiring you provision the hardware. This can be used to identify potential workloads/systems that are good candidates for Exadata migration.

2. SPA supports comparing the performance of two similar workloads or STSs – this functionality is useful when you have mechanisms such as load testing scripts or Oracle Application Testing Suite that can be used to test system changes. By capturing the workload in to two different STSs (for before and after change runs), one can use SPA to assess the impact of the system change.

3. With Oracle Enterprise Manager Cloud Control 12c, a "one-click" STS transport mechanism can be used to simplify the process of moving STS workloads and tuning artifacts such as SQL Profiles or Plan Baselines between production and test databases.

4. SPA supports testing of databases consolidated through Oracle Multitenant Databases or schema consolidation techniques.

SPA Quick Check is an enhancement to SQL Performance Analyzer introduced in Enterprise Manager Cloud Control 12c Release 3, Plug-in Release 1 and can be used on Oracle database versions 11.2 and above.

SPA Quick Check enables verification of routine DBA operations that affect database performance on production databases with a single click of a button. SPA Quick Check runs SQL Performance Analyzer in an optimized mode that is suitable for running on production systems. In this mode, it uses minimal system resources by first identifying SQL with plan changes and then test-executing only those SQL, thereby consuming only a fraction of system resources. SPA Quick Check further limits unusual resource usage by placing run-time limits for each SQL and uses Resource Manager to ensure that only a limited amount of system resources are utilized by this task. All these optimizations make it possible to run SPA on production databases for routine DBA tasks such as optimizer statistics gathering, init.ora parameter changes, etc.

SPA Quick Check enables DBAs to run quick validations for any system change they are required to make to a database and helps improve quality of service of their mission critical databases.

Metric Extensions

Metric extensions provide you with the ability to extend Oracle's monitoring capabilities to monitor conditions specific to your IT environment. This provides you with a comprehensive view of your environment. Furthermore, metric extensions allow you to simplify your IT organization's operational processes by leveraging Enterprise Manager as the single central monitoring tool for your entire datacenter instead of relying on other monitoring tools to provide this supplementary monitoring.

What are Metric Extensions?

Metric extensions also allow you to create metrics on any target type and customize metric thresholds and collections. Unlike user-defined metrics (used to extend monitoring in previous Enterprise Manager releases), metric extensions allow you to create full-fledged metrics for a multitude of target types, such as:

- Hosts
- Databases
- Fusion Applications
- IBM Websphere

- Oracle Exadata databases and storage servers
- Siebel components
- Oracle Business Intelligence components

You manage metric extensions from the Enterprise Manager Cloud Contol 12c Metric Extensions page. This page lists all metric extensions in addition to allowing you to create, edit, import/export, and deploy metric extensions.

The cornerstone of the metric extension is the Oracle Integration Adapter. Adapters provide a means to gather data about targets using specific protocols. Adapter availability depends on the target type your metric extension monitors.

How Do Metric Extensions Differ from User-defined Metrics?

In previous releases of Enterprise Manager, user-defined metrics were used to extend monitoring capability in a limited fashion: user-defined metrics could be used to collect point values through execution of OS scripts and a somewhat more complex set of values (one per object) through SQL.
Unlike metric extensions, user-defined metrics have several limitations:

- Limited Integration: If the OS or SQL user-defined metric executed custom scripts, or required atonal dependent files, the user needed to manually transfer these files to the target's file system.

- Limited Application of Query Protocols: OS user-defined metrics cannot model child objects of servers by returning multiple rows from a metric (this capability only exists for SQL user-defined metrics).

- Limited Data Collection: Full-fledged Enterprise Manager metrics can collect multiple pieces of data with a single query and reflect the associated data in alert context. In the case of user-defined metrics, multiple pieces of data can be collected by creating multiple user-defined metrics, however, it is not possible to refer to the related data when alerts are generated because they are collected separately.

- Limited Query Protocols: User-defined metrics can only use the "OS" and "SQL" protocols, unlike metric extensions which can use additional protocols such as SNMP and JMX.

- Limited Target Application: User-defined metrics only allow OS user-defined metrics against host targets and SQL user-defined metrics against database targets. No other target types are permitted. If, for example, you want to deploy a user-defined metric against WebLogic instances in your environment, you will not be able to do so, making it impossible to associate suspending of monitoring (blackouts) on these targets when servers are undergoing maintenance periods.

Most importantly, the primary difference between metric extensions and user-defined metrics is that, unlike user-defined metrics, metric extensions are full-fledged metrics similar to Enterprise Manager out-of-box metrics. They are handled and exposed in all Enterprise Manager monitoring features as any Enterprise Manager-provided metric and will automatically apply to any new features introduced.

Metric Extension Lifecycle

Developing a metric extension involves the same three phases you would expect from any programmatic customization:

- Developing Your Metric Extension
- Testing Your Metric Extension
- Deploying and Publishing Your Metric Extension

Developing Your Metric Extension

The first step is to define your monitoring requirements. This includes deciding the target type, what data needs to be collected, what mechanism (adapter) can be used to collect that data, and if elevated credentials are required. After making these decisions, you are ready to begin developing your metric extension. Enterprise Manager provides an intuitive user interface to guide you through the creation process.

The metric extension wizard allows you to develop and refine your metric extension in a completely editable format. And more importantly, allows you to interactively test your metric extension against selected targets without having first to deploy the extension to a dedicated test environment. The Test page allows you to run real-time metric evaluations to ensure there are no syntactical errors in your script or metric extension definition.

When you have completed working on your metric extension, you can click Finish to exit the wizard. The newly created metric extension appears in the Metric Extension Library where you can edit can be opened for further editing or saved as a deployable draft that can be tested against multiple targets.

Testing Your Metric Extension

Once your metric extension returns the expected data during real-time target testing, you are ready to test its robustness and actual behavior in Enterprise Manager by deploying it against targets and start collecting data. At this point, the metric extension is still private (only the developer can deploy to targets), but is identical to Oracle out-of-box metrics behavior wise. This step involves selecting your editable metric extension in the library and generating a deployable draft.

You can now deploy the metric extension to actual targets by going through the "Deploy To Targets…" action. After target deployment, you can review the metric data returned and test alert notifications. As mentioned previously, you will not be able to edit the metric extension once a deployable draft is created: You must create a new version of the metric extension.

Deploying Your Metric Extension

After rigorous testing through multiple metric extension versions and target deployments, your metric extension is ready for deployment to your production environment. Until this point, your metric extension is only viewable by you, the metric extension creator. To make it accessible to all Enterprise Manager administrators, it must be published.

Now that your metric extension has been made public, your metric extension can be deployed to intended production targets. If you are monitoring a small number of targets, you can select the Deploy To Targets menu option and add targets one at a time. For large numbers of targets, you deploy metric extensions to targets using monitoring templates. An extension is added to a monitoring

template in the same way a full-fledged metric is added. The monitoring template is then deployed to the targets.

Example:

One good example of a useful metric extension is one that can monitor for SQL that runs too long or uses too much CPU. Of course, what is too long or uses too much CPU is very application and database specific.

SQL Monitor, introduced in Oracle Database 11g, gives a nice graphical view of queries that run longer than 5 seconds or that use parallelism. The data for SQL Monitor can be found in the views GV$SQL_MONITOR and V$SQL_MONITOR. A metric extension can be used to monitor those views for long running SQL. An example SQL that does that is:

```
WITH SQLM as
(select sql_id,sql_exec_start,sql_exec_id
      ,MAX(M.user#)                                        as UserNum
      ,MAX(M.username)                                     as UserName
      ,MAX(NVL(PX_QCInst_ID,M.inst_id))                    as ExecInst
      ,MAX(NVL(PX_QCsid,M.sid))                             as ExecSid
      ,MAX(NVL(PX_QCsid,M.session_serial#))              as ExecSerial
      ,MAX(CASE PX_QCsid WHEN null THEN null ELSE M.status END) as Status
      ,DECODE(count(distinct px_server#),0,'SERIAL','PARALLEL') as
PQ_SERIAL
      ,COUNT(DISTINCT M.inst_id)                 as instances
      ,MAX(NVL(PX_MAXDOP,1))                      as MaxDOP
      ,MAX(last_refresh_time)                     as last_refresh_time
      ,SUM(elapsed_time/1000000)                  as DBtimeSecs
      ,ROUND((MAX(last_refresh_time)-MIN(sql_exec_start))*24*60*60,1)   as
ExecElapsedSecs
      ,ROUND((MAX(last_refresh_time)-MIN(sql_exec_start))*24*60*60 +
SUM(NVL(queuing_time,0)/1000000),1) as TotalElapsedSecs
      ,ROUND(SUM(NVL(queuing_time,0)/1000000),1)     as QueuingSecs
      ,SUM(cpu_time/1000000)                         as CPUsecs
      ,SUM(user_io_wait_time/1000000)                as IOsecs
,SUM((application_wait_time+concurrency_wait_time+cluster_wait_time)/100000
0) as WaitSecs
      ,SUM((plsql_exec_time+java_exec_time)/1000000) as JavaPLSQLsecs
      ,SUM(buffer_gets)                      as BuffGets
      ,SUM(disk_reads)                       as DiskReads
      ,SUM(direct_writes)                    as DirectWrites
 from
      gv$sql_monitor M
group by sql_exec_id,sql_exec_start,sql_id
)
select sql_id               as SQL_ID
      ,ROUND(SUM(CPUSecs))  as TotalCPUSecs
      ,SUM(ExecElapsedSecs) as TotalElapsedSecs
      ,SUM(BuffGets) as TotalBuffGets
      ,SUM(DiskReads) as TotalDiskReads
      ,MAX(UserName)  as SampleUser
```

```
      ,COUNT(*)       as NumExecs
from
      SQLM
where
    (status like 'EXECUTING%'                   -- currently executing or
     OR (status like 'DONE%' AND last_refresh_time > SYSDATE - 15/(24*60))
-- finished last 15 minutes?
    )
group by sql_id
having upper(MAX(Username)) <> 'TOPSQL'
```

This metric extension can be implemented with the thresholds that are appropriate for your environment.

Contact address:

Kurt Engeleiter
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

Phone: (650) 506-0700
Email kengeleiter@gmail.com