Oracle's SQL Performance Analyzer

Khaled Yagoub, Pete Belknap, Benoit Dageville, Karl Dias, Shantanu Joshi, and Hailing Yu
Oracle USA
{khaled.yagoub, pete.belknap, benoit.dageville, karl.dias, shantanu.joshi, hailing.yu}@oracle.com

Abstract

We present the SQL Performance Analyzer, a novel approach in Oracle Database 11g to testing database changes, such as upgrades, parameter changes, schema changes, and gathering optimizer statistics. The SQL Performance Analyzer offers a comprehensive solution to enable users to forecast and analyze how a system change will impact SQL query plans and run time performance, so they can tune their system before they make the change in production. The SQL Performance Analyzer identifies potential problems that may occur and makes suggestions for avoiding any SQL performance degradation. It provides quantitative estimates of the system's performance in the new environment with high confidence and performs a comparative analysis of the response time of the SQL workload thus allowing for an easy assessment of the change. In this paper we describe the architecture of the SQL Performance Analyzer, its usage model, and its integration points with other Oracle database components to form an end-to-end change management solution.

1 Introduction

The past decade has witnessed significant advances in self-managing database technology. The major emphasis of these works [5, 1, 7] has been monitoring a currently running database system for performance regressions, diagnosing any existing performance problems, and suggesting solutions to improve such regressions. While this provides a very effective and complete solution to automatically manage database systems, there is an important aspect of query performance regressions that has been largely overlooked in the database literature: *testing the performance impact of a planned change.* In other words, how well do database systems help administrators prepare for and cope with changes?

System changes could range from simple ones like a new value for a database parameter or the addition of a new index structure to more complex changes like migrating to a newer version of the database or upgrading hardware. Since such changes are inevitable and even the smallest change to the system could have an adverse effect on the performance of certain queries, this is an extremely important problem. Since SQL performance issues are inherently unpredictable, a statement-centric solution makes sense. Users administering critical database systems need a solution to predict the negative effects of a change and take measures to avoid them. Problems left to be discovered on a live system cost the enterprise precious time and resources.

In this paper, we describe the Oracle SQL Performance Analyzer (SPA), which is our solution to the problem of controlling the impact of system changes on query performance. SPA completely automates the manual and

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

time-consuming process of testing the impact of change on potentially large SQL workloads. SPA provides a granular view of the impact of changes on SQL execution plans by executing the SQL statements in isolation before and after a change. Then it compares the SQL execution result before and after the change, and generates a report highlighting the improved and regressed SQL statements and giving precise measurements of their performance impact. Regressed statements are presented with recommendations to remedy their performance.

There have also been some efforts in the industry to address the problem of measuring performance impact caused by system changes. The Quest Plan Change Analyzer [6] relies on the Oracle explain plan command for retrieving the query plans of a set of SQL statements before and after making the desired change and then compares them. While the query plan is often a fair indicator of the actual execution cost of a SQL statement, it may not be very accurate in several situations when there is really no substitute to actually executing the SQL statement to determine its cost. Moreover, unlike SPA, the Quest Plan Change Analyzer does not consider the frequency of execution of SQL statements in a workload while computing performance impact, leading to inaccurate estimates. SPA executes each SQL before and after a change and presents SQL statements ordered by the magnitude of their change on the overall workload performance. For very large workloads, users may not have time to examine each change one by one, so separating the meaningful changes from the rest is very useful.

Hewlett Packard's LoadRunner [8] and Oracle's Database Replay [2] are two more examples of products for evaluating the impact of change on a system. However, these two differ from SPA by providing a complete system workload with timing and concurrency characteristics to a test system. In contrast, SPA computes the performance impact of a change, at the granularity of an individual SQL statement. In this context, SPA is analogous to unit-testing tools while LoadRunner and Database Replay are similar to stress-testing tools.

2 Common Usage Scenarios

SPA can be used to analyze the performance impact of a variety of system changes that can affect the performance of SQL statements. Examples of common system changes include:

- **Database upgrades including patch deployments**: Usually, database administrators (DBAs) are reluctant to upgrade to a new release of the database despite the promising new capabilities the new release offers. This is mainly because they know from past experience that any major release involves significant changes in the database's internal components, which may directly affect SQL performance.

- **Database initialization parameter changes**: The value of a specific parameter can be changed to improve performance, but it may produce unexpected results because the system constraints may change.

- **Schema changes**: Changes such as creating new indexes are intended to improve SQL performance, but they may have adverse effects on certain SQL statements.

- **Optimizer statistics refresh**: Gathering new statistics for database objects whose statistics are stale or missing can cause the optimizer to generate new execution plans. In this case, DBAs can use SPA to assess the benefit of gathering statistics.

- **Implementation of tuning recommendations**: Accepting tuning recommendations from an advisor such as Oracle's SQL Tuning Advisor [5], may require users to test the effect of the recommendations before implementing them.

- **Changes to operating systems and hardware**: Changes, such as installing a new operating system, adding more CPUs, or moving to Oracle Real Application Clusters may also have a significant effect on SQL performance.

3 SQL Performance Analyzer Architecture

Figure 1 illustrates the high level components of the SPA and their interactions with each other.

SPA takes a SQL workload as an input in the format of a SQL tuning set (see Sec. 3.1), executes every statement in the tuning set before and after making the planned change, compares the results of the two executions, and then produces a rich graphical report highlighting the impact of the change at both the SQL workload and individual SQL statement level. SPA is integrated with the optimizer's SQL Plan Management facility and the SQL Tuning Advisor (see Sec. 3.6 and 3.7) to provide support for fixing any regressions that might be caused by the change.

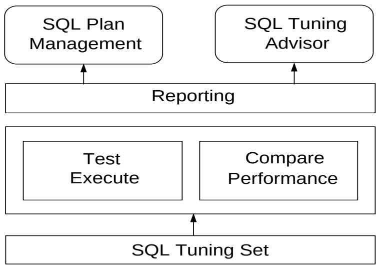

Figure 1: SPA Architecture

3.1 SQL Tuning Set

The SQL tuning set is a database object that provides a complete facility for DBAs to easily manage SQL workload information. A SQL tuning set can be used to capture and persistently store user or application-issued SQL statements along with their execution context, including the text of the SQL, parsing schema under which the SQL statement can be compiled, real bind values used to execute the SQL statement, as well as its execution plans and execution statistics, such as the number of times the SQL statement was executed.

A SQL tuning set can be populated from different SQL sources, including the cursor cache, Automatic Workload Repository (AWR) [5], existing SQL tuning sets, or custom SQL statements provided by the user. SQL tuning sets are transportable across databases and can be exported from one system to another, allowing for the transfer of SQL workloads between databases for remote performance diagnostics and tuning.

3.2 Test-execute

We believe the best way to assess the impact of a change on the performance of a SQL statement is to execute the statement before and after the change and then check if its execution time has regressed or improved. SPA test-executes SQL statements in a SQL tuning set, collects their associated execution statistics and compares them with a previous run of the same statements.

SPA employs an internal SQL service called test-execute to run SQL statements. Test-execute takes as input the text of the SQL statement to execute, actual bind values used on the production system, and a schema name to use to compile the SQL. It then performs a mock execution of the SQL statement with the goal of gathering the SQL execution plan and runtime statistics required for performance comparison. Runtime statistics include elapsed time, CPU time, I/O time, buffer gets, disk reads, disk writes, and row count. During test-execute, the SQL is executed and the produced rows are fetched until the last row in the result set, but never returned to the caller. All rows will be blocked to avoid any side effect, particularly when testing DML and DDL statements. In order to avoid updating the database state, test-execute runs only the query parts of DML and DDL statements, testing the portion of the SQL that is the most vulnerable to change.

SPA executes SQL statements once, one at a time, and in isolation from each other without regard to their initial order of execution and concurrency. This ensures that SPA performs a repeatable experiment whose results can accurately be presented on a per-SQL basis, greatly simplifying the task of interpreting the results.

Explain Plan Option: This option can be used to retrieve only the execution plans for the SQL statements before and after a change and then determine the impact of the change on the structure of the plans. This option is far cheaper than actually executing the statements.

Note that SPA still uses test-execute, but stops it after compilation of the statement to return its execution plan, which is exactly the same execution plan the optimizer would choose, had the SQL been executed with user specified bind values.

Remote Test-execute: SPA also provides the ability to perform test-execute on a remote database using database links. For example, assume that the user is upgrading from Oracle 10.2 to Oracle 11.1 and already has an 11.1 test system set up. She can use SPA on the 11.1 system to first remotely test-execute all SQL statements on the 10.2 system. Next, she can perform another test-execute, but on the local system and then compare the two sets of execution plans and runtime statistics.

To perform a remote test-execute, SPA automatically establishes a connection to the remote database using a database link specified by the user, executes the SQL statements on that database, collects the execution statistics and plan for each statement, and then stores them back in the local database for analysis and comparison.

Time Limit: To control the time spent while processing a SQL tuning set, SPA allows users to specify two time limits for test-execute: 1) A global time limit which represents the maximum duration for processing a SQL tuning set. This time limit is important, particularly, when using a large SQL workload. 2) A per-SQL time limit which is the maximum duration for the processing of a single SQL. The per-SQL time limit is used to control runaway queries. When set by the user, the same time limit applies to every SQL in the SQL tuning set.

3.3 Compare Performance

This SPA module is responsible for comparing the performance of the SQL workload before and after a change, and calculating the impact of the change on the SQL workload.

SQL Trial: The output of test-executing a SQL tuning set, i.e., the resulting execution plans and runtime statistics, are stored in the database in a container called a SQL trial. A SQL trial represents a particular experiment or scenario when testing a given change. It encapsulates the performance of a SQL workload under particular conditions of the system.

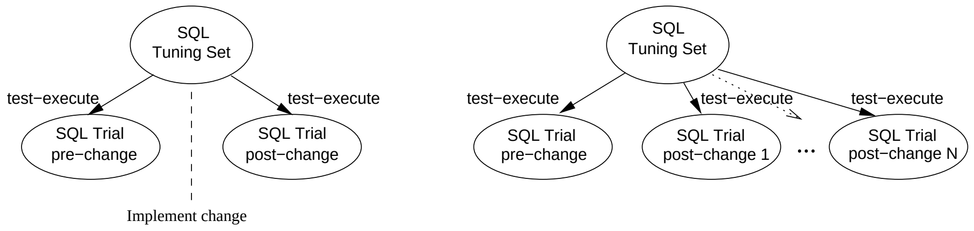

Figure 2: SPA with (a) Two SQL Trials, (b) Multiple SQL Trials

As the above diagram shows, the user can create any number of SQL trials, where each trial corresponds to the SQL workload performance data under a different change, and compare any two trials. All trials will reside in the database, thus forming a history of all testing experiments conducted by the user for a SQL workload. This is a very useful feature of SPA as it allows users to keep track of changes and perform historical performance analysis. SPA's iterative usage model is a recognition to the fact that the nature of system testing is one of one change leading to another, with each being tested in isolation until a steady state is reached.

Performance Comparison: Once performance data has been gathered under each SQL trial, the performance comparison module analyzes the differences between two trials and unmasks the SQL statements that are impacted by the tested change. The compare module measures the impact of the change on both the overall performance of the SQL workload as well as on each individual SQL statement. By default, SPA uses the elapsed time as a metric for comparison. The user can also choose from a variety of available SQL runtime statistics, including SQL CPU time, I/O time, buffer gets, disk reads, disk writes, or any combination of them as an expression (e.g., cpu_time + 10*buffer_gets). The module also compares the execution plans' structural changes of SQL between the two trials.

Change Impact Calculation: Change impact is a measure of how a system change affects the performance of a SQL statement. SPA calculates the change impact based on the difference in resource consumption across two trials of the SQL workload as follows:

$$ciw = \frac{\sum_i e_{bi} f_i - \sum_i e_{ai} f_i}{\sum_i e_{bi} f_i} \qquad cis_i = \frac{e_{bi} - e_{ai}}{e_{bi}} \qquad cisw_i = \frac{f_i(e_{bi} - e_{ai})}{\sum_i e_{bi} f_i} \tag{1}$$

ciw: change impact on the overall performance of the workload.
cis_i: change impact on individual SQL in the workload.
$cisw_i$: impact of a SQL performance change on the overall performance of the workload.
f_i: execution frequency, i.e., number of executions, of a given SQL captured in SQL tuning set.
e_{bi}: execution metric of a SQL single test-execution from the before change SQL trial.
e_{ai}: execution metric of a SQL single test-execution from the after change SQL trial.

These measurements are presented to the user through the SPA report. As a general rule, negative values indicate regressions, while positive values indicate improvements in performance.

The SQL execution frequency is used by SPA to weight the importance of each SQL statement in the workload. This allows users to correctly determine the impact on long running SQL statements that are executed only a few times as well as statements which are very fast, but repetitively executed.

3.4 Reporting

When the performance comparison and analysis are complete, all resulting data are written into the database. The end user can then review the analysis findings produced by SPA by either directly querying the exposed schema or simply requesting the analysis report from SPA.

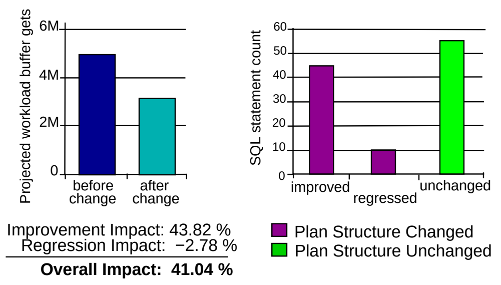

Figure 3: Example of a Partial SPA Report

The SPA report is divided into two main sections: Analysis Summary and Analysis Details. The summary section gives statistics about the overall change in performance of the SQL workload and points out the SQL statements that are impacted by the change. The detail section has an entry for every SQL statement in the SQL workload with detailed information about the SQL as well as a side-by-side comparison of the SQL runtime statistics and execution plans from the trials used in the comparison. In the report SQL statements are ordered by their change impact on the SQL workload performance.

As depicted in Figure 3, the report shows graphically the overall value of an *buffer gets* before and after making the system change, along with a second graph for the count of SQL statements whose performance improves, regresses or remains unchanged as a result of the change. Both these graphs have drill-down capabilities to view details at individual SQL statement level. The example above indicates that overall, the workload performance improved by 41.04% even though it experienced some regressions as shown by the impact of -2.78%.

3.5 SQL Plan Management

If the comparison of two SQL trials shows some SQL statements with regressed performance, SPA will recommend creation of plan baselines[1] [3] for the subset of regressed SQL using execution plans from the *first* SQL trial. This ensures that the optimizer will always use those plans for future executions of this subset of SQL statements preserving their performance, regardless of changes occurring in the system.

3.6 SQL Tuning Advisor

SPA will also recommend SQL tuning advisor [4] to fix performance problems. The SQL tuning advisor analyzes each regressed SQL statement with the goal of finding a SQL profile that will counteract the negative impact of the change. SQL profiling attempts to discover the root cause of a SQL performance problem by understanding the complex relationships in the data relevant to the execution of the SQL statement.

For statements whose performance could not be improved by the tuning advisor, the user can create plan baselines with SPA to ensure that their performance will be no worse than what it used to be before the change.

4 Usage Model

Oracle Enterprise Manager provides a graphical interface that guides a user through each of the steps mentioned in this section. We assume that a test system is available and that it resembles the production system as closely as possible. However, users can run SPA directly on the production system if, for example, they cannot afford a test system or if they have a sufficient time window to test their changes on production.[2]

4.1 Basic Testing Workflow

As Figure 4 illustrates, the testing process using SPA has the following steps:

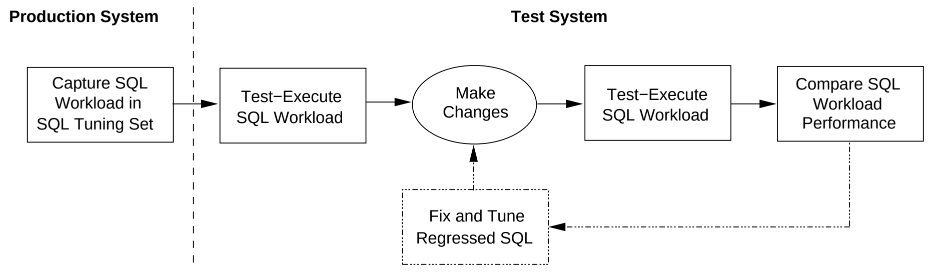
Production System
Test System
Capture SQL Workload in SQL Tuning Set
Test−Execute SQL Workload
Make Changes
Test−Execute SQL Workload
Compare SQL Workload Performance
Fix and Tune Regressed SQL

Figure 4: SPA Basic Testing Workflow

1. Capture SQL Workload: Before running SPA, users have to capture on the production system a set of SQL statements that represent the SQL workload they intend to analyze. The higher the number of SQL statements captured in the workload, the more accurate the prediction of performance changes will be. The set of SQL statements is captured and stored in a SQL tuning set. SQL tuning set provides an incremental SQL workload capture facility that enables the capture of the entire system SQL workload with minimal performance overhead. Incremental capture works by repeatedly polling the cache of currently executing SQL statements over a period of time.

[1]A plan baseline is an optimizer feature that guarantees stable performance in the face of runtime changes by maintaining a history of past execution plans for repeatable statements.

[2]Using a test system is not mandatory, but recommended since SPA test executes SQL before and after the change and this could be very resource-intensive depending on the complexity and size of the workload.

2. Transport SQL Tuning Set: After creating the SQL tuning set with the appropriate SQL workload, it is exported from the production system and imported into a test system where the system change under consideration will be tested. This can be achieved by using SQL tuning set export/import capabilities.

3. Test-execute SQL Before Change: After the SQL workload is captured and the SQL tuning set transported to the test system, SPA can be used to build the *pre-change* SQL Trial. SPA test-executes the SQL tuning set and produces execution plans and runtime statistics for each statement in the tuning set. SPA can also be run to generate SQL execution plans only, i.e., without collecting execution statistics. This technique reduces the time of SPA execution, but the results of the comparison analysis are not as complete because, without executing the SQL, it is impossible to make accurate predictions about its impact on system resource statistics.

4. Perform Change: After the pre-change trial is built, the system change to test can be implemented on the test system. This change can be any kind of change that might impact the performance of SQL statements such as a database upgrade, new index creation, initialization parameter changes, optimizer statistics refresh, etc.

5. Test-execute SQL After Change: After implementing the planned change, SPA can be invoked again to re-execute the SQL statements and produce execution plans and execution statistics for each SQL statement, a second time. This execution result represents the *post-change* trial that SPA uses to compare against the *pre-change* SQL trial. The user can also combine the explain plan option with test-execute to speed up the testing process. For example, she can start by running SPA using the explain option to retrieve the plans for all SQL in the workload and then execute only the subset of SQL whose plans changed to verify whether those plans improved or regressed.

6. Compare Performance: SPA uses the metric specified by the user and compares the performance data of SQL statements in the pre-change SQL trial to the post-change SQL trial. Finally, it produces a report identifying any changes in execution plan structures or performance of the SQL statements. The SPA analysis report explains how the tested change impacts the performance of a SQL workload and what actions can remedy the uncovered regressions.

It is important to note that neither the before nor the after SQL trial gains an undue advantage from certain system conditions such as cached data. In this case, the user can perform a dummy test execute trial to guarantee consistent caching of data across the two trials or simply use a comparison metric that is not dependent on caching such as, CPU time or buffer gets.

7. Re-iterate: If the performance comparison reveals regressed SQL statements, then the user can make further changes to fix the problematic SQL by creating SQL plan baselines or SQL profiles. The testing process can be repeated until the user has a clear understanding of the impact of the change and the corrective actions to improve the potential performance regressions. The user can then be confident to permanently make the change on production and implement the tuning actions even before the performance degradations occur.

4.2 Parameter Change Workflow

In addition to the basic testing workflow, SPA provides a predefined workflow to test database parameter alterations. This workflow enables the user to test the performance effect on a SQL tuning set when varying the value of an environment initialization parameter. Given a SQL tuning set and a comparison metric, SPA automatically creates two SQL trials and compares them. The first trial captures SQL performance with the initialization parameter set to the original value, whereas the second trial uses the new value of the parameter.

5 Conclusion

Database changes happen all the time and affect SQL performance. Therefore, one of the most important tasks for DBAs is to assess the potential impact of any changes to the database environment on SQL performance. This is a very challenging task because it is almost impossible to predict the impact of changes on SQL performance before actually implementing them in the production system. Building a thorough test bed with the ability to make reliable predictions about the impact of such changes has historically been beyond the reach of most system administrators.

In this paper, we have described SQL Performance Analyzer, which was introduced in Oracle 11g. SPA gives users the ability to measure the impact of system changes on the performance of SQL statements and fix any potential regressions before they happen in production. SPA helps DBAs build and compare different versions of SQL execution plans and runtime statistics, and then suggests tuning recommendations to overcome potential performance problems.

We have discussed the primary end user of SPA as a production DBA, but it can also be used by other types of users, such as QA testers and application developers. With SPA, DBAs have the necessary information to determine what performance changes may occur in a SQL workload and what corrective actions to undertake to fix regressions. At the same time, QA teams can use it to identify, investigate, and solve performance issues before they occur during a new application deployment. Likewise, application developers can use SPA to measure and control the risk of performance changes throughout their application's life cycle. All of these users can benefit from a comprehensive product with the ability to measure the performance impact of a change to a real SQL workload. As long as enterprises continue to expand and adapt to new environments, change will be a constant in database systems. By forecasting the impact of changes before they are implemented in production, we believe that tools like SPA eanble DBAs to clearly understand the performance ramifications of system changes and take corrective actions to avoid any potential degradations.

References

[1] S. Agrawal, N. Bruno, S. Chaudhuri, and V. Narasayya. Autoadmin: Self-tuning Database Systems Technology. *IEEE Data Eng. Bull.*, 29(3):7–15, 2006.

[2] J. Athreya and M. Minhas. Oracle Database 11g Real Application Testing Overview. Technical report, Oracle, USA, http://www.oracle.com, 2007.

[3] M. Colgan. SQL Plan Management in Oracle Database 11g. Technical report, Oracle, USA, http://www.oracle.com, 2007.

[4] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic SQL Tuning in Oracle 10g. In *VLDB*, pages 1098–1109, 2004.

[5] B. Dageville and K. Dias. Oracle's Self-Tuning Architecture and Solutions. *IEEE Data Eng. Bull.*, 29(3):24–31, 2006.

[6] C. Fernandez and J. Leslie. Predicting and Preventing Performance Bottlenecks in Oracle 10g. Technical report, Quest Software, http://www.quest.com, 2005.

[7] S. Lightstone, G. Lohman, P. Haas, V. Markl, J. Rao, A. Storm, M. Surendra, and D. Zilio. Making DB2 Products Self-Managing: Strategies and Experiences. *IEEE Data Eng. Bull.*, 29(3):16–23, 2006.

[8] H. Packard. LoadRunner. Technical report, http://www.hp.com, 2007.