Automatic Performance Diagnosis and Tuning in Oracle

Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, Graham Wood

Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065, USA
{kdias, mramache, ushaft, veeve, gwood}@oracle.com

Abstract

Performance tuning in modern database systems requires a lot of expertise, is very time consuming and often misdirected. Tuning attempts often lack a methodology that has a holistic view of the database. The absence of historical diagnostic information to investigate performance issues at first occurrence exacerbates the whole tuning process often requiring that problems be reproduced before they can be correctly diagnosed.

In this paper we describe how Oracle overcomes these challenges and provides a way to perform automatic performance diagnosis and tuning. We define a new measure called 'Database Time' that provides a common currency to gauge the performance impact of any resource or activity in the database. We explain how the Automatic Database Diagnostic Monitor (ADDM) automatically diagnoses the bottlenecks affecting the total database throughput and provides actionable recommendations to alleviate them. We also describe the types of performance measurements that are required to perform an ADDM analysis. Finally we show how ADDM plays a central role within Oracle 10g's manageability framework to self-manage a database and provide a comprehensive tuning solution.

1. Introduction

With the advent of web business, system performance is more important to businesses than ever before. A poorly performing web site is just as bad as one that is actually down, because users get frustrated and are unlikely to return. Databases are an integral part of these high profile systems. With the decreasing costs of computer hardware, businesses are building more and bigger databases, and the database administrators (DBAs) are being asked to take on more and larger databases.

Database systems typically have a plethora of measurements and statistics available about their operation. Often many individual components maintain separate sets of measurements and it can be hard to get an overall view of what is happening in the database. The many sets of measurements can lead to data overload for the DBA who is tasked with analysing a problem.

Identification of the root cause of a performance problem is not easy [WE02, CH00, HE97, BR94]. It is not uncommon for DBAs to spend large amounts of time and resources fixing performance *symptoms*, only to find that this had marginal effect on system performance. Often a symptom is treated mainly because the DBA has seen that symptom before and knows how to treat it. Lack of a holistic view of the database leads to incorrect diagnosis and misdirected tuning efforts resulting in over-configured systems increasing the total cost of ownership [HU02, CH00, GA96].

Even when 'correct' analysis is performed it is often found that the available data stops short of what is required to fully diagnose the root cause of the problem and in order to complete the diagnosis the problem must be reproduced. Reproducing problems is sometimes non-trivial and can range from simply asking a user to perform an operation again, to requiring the building of a copy of the database, which may take weeks to set-up. Many times performance issues go unresolved because it is not regarded as cost effective to spend the time and resources to reproduce the problem. Often, the issue is dropped and the DBA hopes that the problem will not reoccur. In most cases, the mechanisms provided to capture detailed information required to be able to fully analyse a problem are not designed to be non-intrusive and have a significant overhead. Sometimes tuning experts or 'gurus' are called in to try and resolve a problem. Specialist skills and expertise are required because there are many areas where value judgments and rules-of-thumb are involved.

Proceedings of the 2005 CIDR Conference.

One of the things that makes performance tuning difficult is that the performance of different components of the database is often measured using unrelated metrics like buffer hit-ratio, transactions per second and average I/O latency. Such unrelated metrics make it infeasible to weigh the performance impact of one component of the database with the impact of others components. Tuning actions that are solely based on such metrics have a tendency either to make the DBAs believe that the database needs excessive hardware or to make just one component of the database perform better with negligible effect on the total database throughput.

Keeping these limitations in mind, we built the Automatic Database Diagnostic Monitor (ADDM) in Oracle 10g. ADDM automates the entire process of diagnosing performance issues and suggests relevant tuning recommendations with the primary objective of maximizing the total database throughput.

Modern database systems have complicated interactions between their sub-components and have the ability to work with a variety of applications. This results in a very large list of potential performance issues such automatic diagnosis solutions could identify. Also, as new database technologies and applications are invented and older ones are obsoleted, it is pivotal that automatic diagnostic and tuning solutions can easily be adapted to accommodate such changes.

The objectives of an automatic performance diagnosis and tuning solution can be summarized as follows:

- Should posses a holistic view of the database and understand the interactions between various database components.
- Should be capable of distinguishing symptoms from the actual root-cause of performance bottlenecks.
- Should provide mechanisms to diagnose performance issues on their first occurrence.
- Should easily keep up with changing technologies.

The remainder of the paper is organized as follows: We discuss related work in both academic research and commercially available databases in Section 2. We define a new measure, Database Time, and describe in detail our methodology to solve this problem in Section 3. We discuss the various measurements that were required to implement our solution in Section 4. We extend our definition of Database Time to other models in Section 5 and explain how ADDM plays a central role in Oracle 10g's self-managing framework in Section 6. We present the experiments we used to validate our implementation and their results in Section 7. Finally, we summarize our conclusions in Section 8.

2. Related Work

The COMFORT project used an online feedback control loop to solve individual tuning issues like load control for locking, dynamic data placement among others [WE02, WE94]. Automated tuning systems have been proposed [HE97] that employ a feedback control mechanism layered on top of a target system. A different school of thought suggests that current database systems have too many features, unpredictable performance and very low "gain/pain ratio", and RISC-type simplification of database functionality is crucial to achieve automatic tuning [WE02, CH00].

Oracle's previous solution for performance tuning was Statspack [ORP8, ORP9, ORSP]. Statspack is a tool that takes snapshots of database performance statistics and generates reports across a pair of snapshots. Though Statspack reports reduced the time it took for DBAs to diagnose performance issues [ORSP2], it still left the interpretation of the report and the actual root-cause identification to the DBAs. Oracle 9i has other self-optimizing features like dynamic runtime query optimization [ORQ9], self-configuring features for space and memory management [DA02, ORM9, EL03]. Refer to Section 6 for a description of various Oracle 10g's manageability features like "SQL Tuning Advisor" and "Segment Advisor". IBM DB2 version 8.1 has features like "Health Center" for database monitoring and "Configuration Assistant" to assist DBAs in database configuration [IBP8, IBG8]. Similarly, SQL Server 2000 provides performance tools like "Index Tuning Wizard" and "SQL Query Analyzer" to help the DBA achieve individual performance goals and has monitoring tools like "System Monitor objects" [CH97, MSPD].

Commercially available databases have features to manage some of their sub-components automatically, mechanisms to actively monitor database performance, and tools that make it easier for the DBA to achieve some specific performance goals. However, none of them (until Oracle 10g) have a performance diagnosis and tuning solution that automatically identifies the root-causes affecting total database throughput.

3. Automatic Performance Diagnosis

Performance of various components of the database are measured using different metrics. For example, the efficiency of the data-block buffer cache is expressed as a percentage in buffer hit-ratio; the I/O subsystem is measured using average read and write latencies; the database throughput is measured using transactions-per-second. However, using these metrics, finding the performance impact of a particular component over the total database throughput is extremely hard, if not infeasible. For example, determining by how much would transactions-per-second decrease if the average I/O latency becomes twice as long is not trivial.

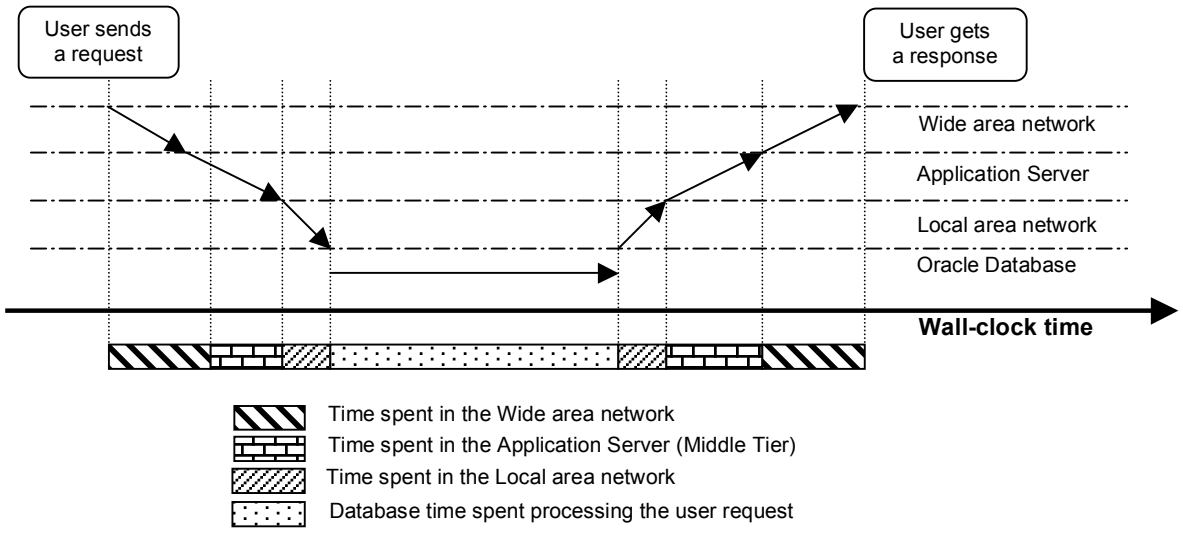


Figure 1: Database time from a single user's perspective.

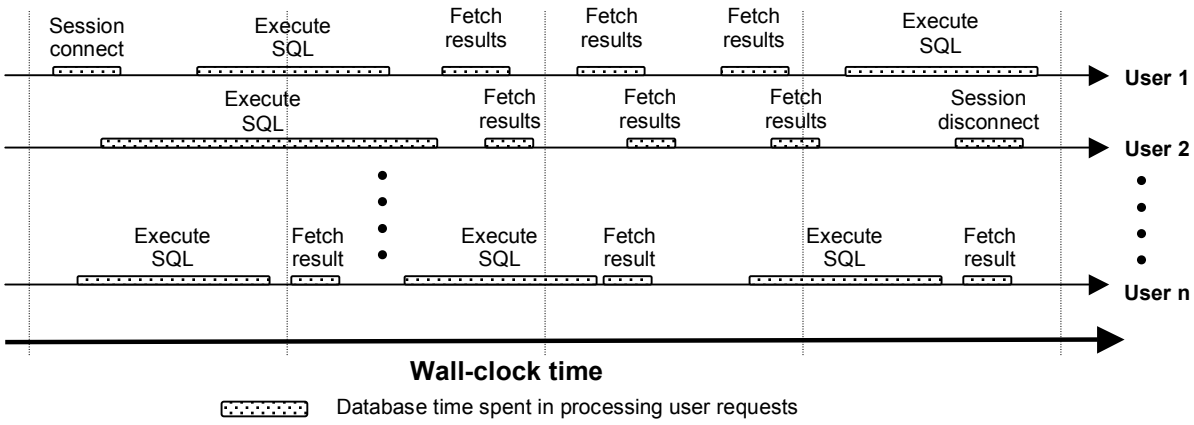


Figure 2: Database time defined as the sum of the time spent in processing all user requests

One of the first objectives of this project was to define a common currency that can be used to measure the performance impact of any component within the database. It then becomes possible to be able to compare the impact of any two database components. For example, one would then be able to compare the performance impact due to an under-sized data-block buffer cache with the performance impact due to a badly written SQL statement. For this purpose, we define a measure called 'Database Time' that would serve as a common currency.

3.1 Database Time

Database time is defined as the sum of the time spent inside the database processing user requests. Figure 1 illustrates a simple scenario of a single user submitting a request. User's response time is the time interval between the instant the request is sent and the instant the response is received. The database time involved in that user request is only a portion of that user's response time that is spent inside the database.

Figure 2 illustrates database time as it is summed over multiple users and each user is performing a series of operations resulting in a series of requests to the database. It can be seen from Figure 2 that database time is directly proportional to the number and duration of user requests and can be higher or lower than the corresponding wall-clock time.

Using database time as a measure, one should be able to gauge the performance impact of any entity of the database. For example, the performance impact of an under-sized buffer cache would be measured as the total database time spent in performing additional I/O requests that could have been avoided if the buffer cache was larger.

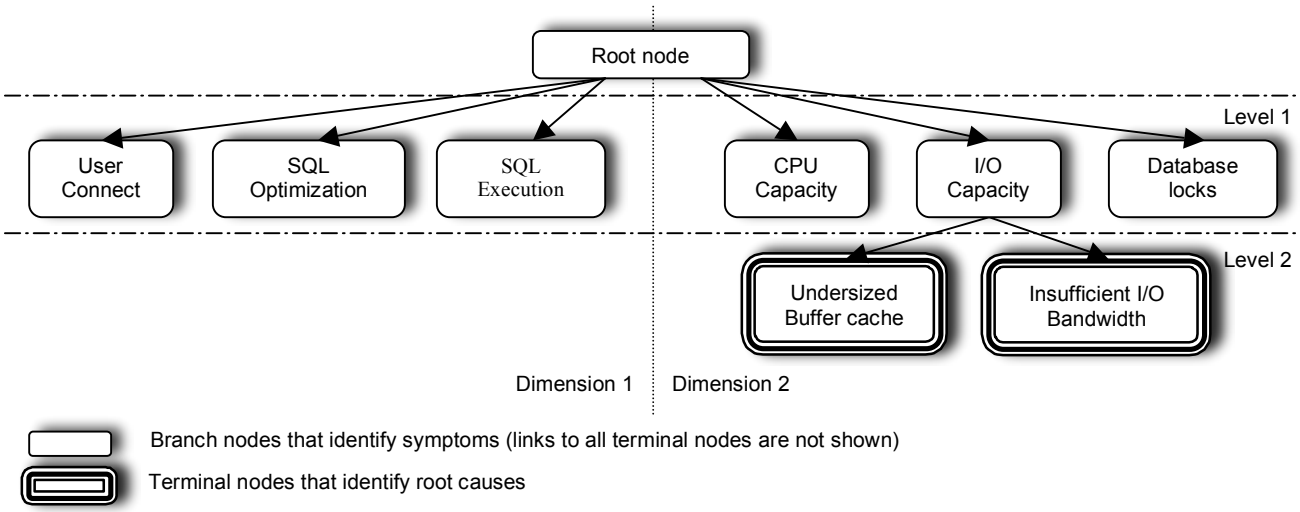


Figure 3: Sample DBTime graph

Database time is simply a measurement of the total amount of work done by the database (analogous to the SI unit 'Joule' used in Physics to measure energy or work done) and the rate at which the database time is consumed is the database load average which is analogous to the OS load average (measured as 'Database Time'/second this unit is analogous to the SI unit 'Watt' or 'Joules/second' used in Physics to measure power). The objective of ADDM is to reduce the amount of database time spent on a given workload, which is analogous to consuming less energy to perform the same task. Identifying the component contributing the most database time is equivalent to finding the single database component that when tuned will provide the greatest benefit. In other words, if we look for ways to process a given set of user requests while consuming the least amount of database time, we are simply looking for the most efficient way to accomplish the given task.

ADDM uses database time to identify database components that require investigation and also to quantify performance bottlenecks.

Database systems often have models or mechanisms, like background processes or parallel queries, in which the definition of the database time is more complex. In Section 5 we discuss database time definitions for these models.

3.2 DBTime-graph and ADDM Methodology

The first step in automatic performance tuning is to correctly identify the causes of performance problems. Only when the root-cause of the performance problem is correctly identified, it is possible to explore effective tuning recommendations to solve or alleviate the issue.

ADDM looks at the database time spent in two independent dimensions.

- The first dimension looks at the database time spent in various phases of processing user requests. This dimension includes categories like 'connecting to the database', 'optimizing SQL statements', and 'executing SQL statements'.
- The second dimension looks at the database time spent using or waiting for various database resources used in processing user requests. The database resources considered in this dimension include both hardware resources, like CPU and I/O devices, and software resources like database locks and application locks.

For a detailed analysis on how we actually measure the database time spent on various categories mentioned here refer to Section 4.

To perform automatic diagnosis, ADDM looks at the database time spent in each category under both these dimensions and drills down into the categories that had consumed significant database time. This two-dimensional drill down process can be represented using a directed-acyclic-graph as shown in Figure 3, which we call the "DBTime-graph".

It should be noted that this DBTime-graph is not a decision tree for a rule-based diagnosis system, where a set of rules is organized in the form of a decision tree and tree is traversed either to find the goal given a particular set of data or to find the data given a particular goal [BE03]. The DBTime-graph has various properties that differentiates itself from rule-based decision trees:

- Each node in this graph looks at the amount of database time consumed by a particular database component or resource.
- All nodes in this graph are gauged with the same measure - database time.

- All the children of a particular node are unconditionally explored whenever the database time spent in that node is significant.
- Database time attributed to a particular node should be completely contained in the database time attributed to each of its parents.

Any node that complies with all the properties above can be added to the DBTime-graph making it easy to maintain with changing technologies, unlike the decision tree of a rule-based diagnosis system [BE03].

Performance problems often attribute database time across many categories in one dimension but not the other. For example, a database with insufficient CPU capacity will slow down all phases involved in processing user requests, which is ADDM's first dimension.

However, it would be evident from the second dimension that the top performance problem affecting the database is insufficient CPU capacity. This two dimensional view of looking where the database time is consumed gives ADDM a very good judgment in zooming in to the more significant performance issues.

ADDM explores this DBTime-graph starting at the root-node and exploring all the children of a node if the database time consumed by the component is significant. Branch nodes in this graph identify the performance impact of what is usually a symptom of some performance bottleneck. Whereas the terminal nodes identify particular root-causes that can explain all the symptoms that were significant along the path in which the terminal node was reached. For example, in Figure 3 the branch node "I/O Capacity" would measure database time spent in all I/O requests, which could be significant due to various bottlenecks. Whenever significant database time was spent in I/O requests all the children of the "I/O Capacity" node would be explored, which are the two terminal nodes in this example. The "Undersized Buffer Cache" node would look for a particular root-cause, which is to see if the data-block buffer cache was undersized causing excessive number of I/O requests. The "Insufficient I/O Bandwidth" node would look for hardware issues that could slow down all I/O requests.

Once a terminal node identifies a root-cause, it measures the impact of the root-cause in database time. It then explores ways that can solve or alleviate the problem identified. It uses the various measurements that are discussed in Section 4 to come up with actionable tuning recommendations. The nodes also estimate the maximum possible database time that could be saved by the suggested tuning recommendations, which need not necessarily be equal to the database time attributed to the root-cause.

In addition to identifying bottlenecks, ADDM also identifies key components that were not experiencing any performance bottlenecks. The idea is to prevent the DBA from tuning components that have marginal effect on the total database throughput.

It is interesting to note that ADDM need not traverse the entire DBTime-graph; it can prune the uninteresting sub-graphs. This can be achieved only because the DBTime-graph is constructed in a way that a node's database time is contained in the database time attributed to its parents. By pruning and not traversing uninteresting sub-graphs, which represent database components that are not consuming significant database time, the cost of an ADDM analysis depends only on the number of actual performance problems that were affecting the database. The cost of the analysis does not depend on the actual load on the database or the number of issues ADDM could potentially diagnose.

The DBTime-graph also makes it very easy for ADDM to adapt to new requirements to look for new root-causes for performance problems without affecting the cost of the analysis. Evidently, removing obsolete terminal nodes is also very simple to do.

At the end of the analysis ADDM reports the top root-causes identified, ranked by the performance impact attributed with each root-cause along with the respective tuning recommendations.

4. Workload Measurements

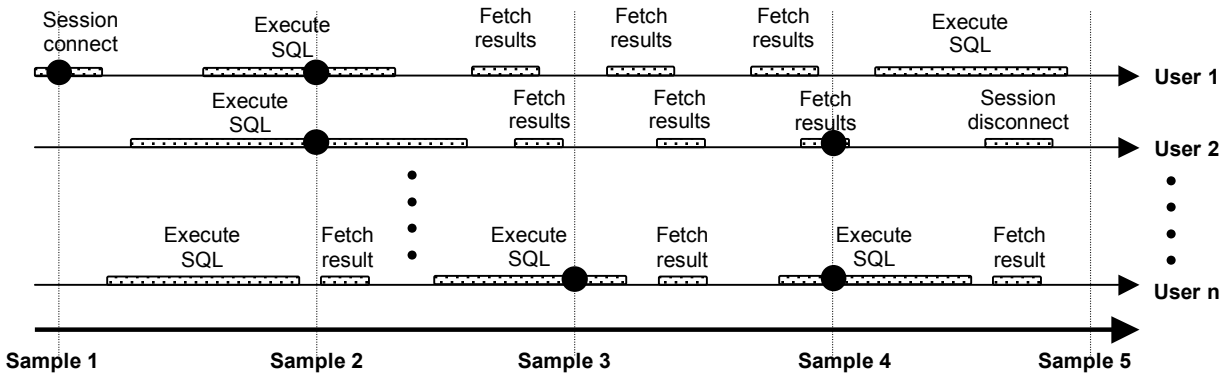
ADDM analysis can only be done if the appropriate data is available. In this section we discuss the performance data that a system should collect to enable an ADDM analysis. In this section we list the requirements imposed by ADDM and describe the types of data we collect for any analysis period. Refer to Section 6 for details about how the data is stored and used in practice.

4.1 Data Collection Requirements

Our first and most important requirement is that we collect all the data ADDM needs for each node in the DBTime-graph. ADDM needs data for the following operations:

- Quantifying the impact in database time of the database components under consideration.
- Finding recommendations for alleviating root-cause problems and estimating the potential benefit of a recommendation in database time.

Our second requirement is the "minimal intrusion principle"; it states that the act of collecting measurements for performance diagnostics should not cause a significant degradation in performance. Obviously, the notion of "significant" is subjective and is determined by business considerations—what kind of server overhead is acceptable for the customer to get the benefit of an ADDM analysis. Any measurement we take has an associated cost. At the very least we need to increment some counter or an in-memory variable. If we need exact timing, we might need to use an operating system call to get the current time. This cost might be



Sample Number	User Number	Operation	Details
1	1	Connect	Using CPU
2	1	Execute	Using CPU
2	2	Execute	Waiting for read I/O of block 5 of Emp table
3	n	Execute	Waiting for lock on block 5 of Emp Table
4	2	Fetch	Using CPU
4	n	Execute	Waiting for CPU

Figure 4: Active Session History Samples

negligible in some cases, but could be prohibitive in others.

In sections 4.2 to 4.5 we describe the different types of data we collect for ADDM analysis. In all cases we describe both the purpose of the data and how it conforms to the minimal intrusion principle.

4.2 Database Time Measurements

The first priority in an ADDM analysis is to establish the main components that consume significant database time. The database server must be instrumented to measure the amount of database time spent in specific operations ADDM is interested in. This measurement is a cumulative non-decreasing function of time. ADDM analysis is always performed over a specific time interval called the "analysis period" (e.g., yesterday between 10am and 11am). ADDM finds how much time was spent on a specific operation by subtracting the measurement for the beginning of the analysis period from the measurement for the end of the analysis period.

We maintain measurements at various granularities so an ADDM analysis can go from the symptoms (e.g., commit operations consumed significant database time) to the root cause (e.g., write I/O to one of the log files was very slow—possibly a hardware issue). We need to maintain measurements at the coarse level—time spent in all commit operations. We also need the finer granularity—time spent in write I/O for each log file.

Direct measurements can only be done on database operations that usually take significant time to finish. If we try to measure the timing of numerous short operations we are likely to violate the minimal intrusion principle. The decision about which operations should be measured should be based on the cost of measurement (i.e., how much time it takes to start and end a timer) and the expected length and quantity of such operations. For example, measuring the total time spent in I/O operations is reasonable because I/O operations take many orders of magnitude more time than starting and ending a timer. On the other hand, measuring the time we spend in the critical section of a commonly used semaphore would be prohibitively expensive. Our solution to capture short duration operations is to use sampling. We use both frequency based sampling, where we sample one operation out of every N occurrences, and time based sampling which is discussed in detail in the next section.

4.3 Active Session History

ADDM often needs a detailed description of the workload to be able to find root-causes of problems and give

effective recommendations. It is not possible to collect a complete system trace of operations since the amount of data is very large and it will significantly affect database performance. Our solution is to provide a time based sample of all activity in the system. We call this sampled data the "Active Session History" (ASH).

ASH is a collection of samples taken at regular time intervals. Each sample contains information about what the database server is doing on behalf of each connected user (a.k.a. "session") at the time of sampling. We only collect data for sessions that are actively using the database during the sample time. If a session is waiting for the next user command, it does not appear in ASH for the specific sample time.

Figure 4 illustrates a workload and the table in that figure shows the resulting ASH samples. The samples in ASH describe the database operation performed by each session in great detail. For example, when a session is waiting for I/O we can tell which database object is read, what portion of it is read. This amount of detail enables ADDM to drill down to very specific root-causes.

The time interval we use for ASH sampling should be at least two orders of magnitude shorter than the time interval used as an analysis period. This ensures that we have at least hundreds of samples of specific session states to analyze. If a specific operation consumes significant database time (say more than a few percents of database time) during the analysis period, there is a high probability that this operation will appear in a significant number of samples in ASH. This enables ADDM to diagnose such operations even if we do not measure them directly. The ASH sampling frequency should be chosen with care. If sampling is too frequent, it might violate the minimal intrusion principle. If sampling is infrequent, ADDM may not be able to find the root-cause of a problem or an effective recommendation, because there may not be enough samples of the specific operation.

ADDM uses ASH for finding root-causes and effective recommendations. For example, suppose we find from exact database time measurements that the chief source of contention in the system is on locks that deal with space allocation to tables. We can use ASH to find finer granularity of events like "which are the top tables that experienced space allocation contention and what are the types of SQL statements used for such contention?" We might be able to determine that a specific table suffers from contention due to INSERT statements and recommend a partitioning scheme to alleviate the problem. While time measurements might be able to lead us to the specific type of locks that cause a problem, or sometimes to the specific table, only ASH allows us to perform a cross analysis with the types of SQL statements that contended for the locks. Without ASH, ADDM could not recommend the specific solution: partition a specific table.

4.4 System Configuration

We also collect data that is not numeric in nature or that can decrease with time. This data is usually about database settings. We need to maintain a full log of changes for this kind of data. Fortunately, database settings do not change very often, so the cost of maintaining a full log of changes does not cost much. This type of data can be crucial to giving recommendations for fixing specific problems. Examples of such data are:

- Size of memory components (like buffer cache)
- Number of CPUs used by the system.
- Special query optimizer settings.

The exact nature of the data depends on the implementation of the database server and is outside the scope of this paper.

4.5 Simulations

Sometimes, estimating the impact of a specific area of the database requires a simulation of various possible alternatives. For example, we might have significant impact of read I/O operations because the buffer cache is under-sized. The database time measurement is the time we spent on read I/O operations. However, to find that the buffer cache is the root-cause we must determine that we spent time reading data blocks that were in the buffer cache at some point in time and were removed to make room for other data blocks. In other words, we need to determine how many read I/O operations could have been saved given an infinite buffer cache. Our solution is to simulate how much database time would be saved when the buffer cache is increased to various sizes and recommend an optimal setting given the resources.

5. Database Time in Other Models

The definition of database time in Section 3.1 relies on a simple computation model. The model is characterized by the following limitations:

- Single Machine. This means that we do not have a distributed system. Instead, we use a single host machine for the database server.
- No Background Activity. This means that all database work is done while the user is waiting for a response.
- No Parallel Computation. Every user connection corresponds to a single thread of computation in the database server. This means that work done on behalf of a user call cannot use more than one CPU at the same time, wait for more than one event at the same time, or wait for an event and use a CPU at the same time.

Most database servers support computation models that use distributed systems, parallel queries and background activity. Therefore, we needed to establish ADDM's goals when each of these assumptions is violated.

5.1 Distributed Systems

The simplest distributed system has identical components. This means that all the machines that are part of the system are identical machines, running the same software, have the same network connections, and use the same types of I/O devices. In this simple case, all we need to do is sum the database time observed on all machines to get the total database time for the distributed system. Every subdivision of database time (e.g., the time spent waiting for I/O) is summed across all machines as well.

It is a lot more complicated when there are differences between machines. In that case we need to find a common currency between machines and not just between different areas in the same server. For example, we cannot consider one second of CPU usage on one machine equivalent to one second of CPU usage on another machines if the two CPUs have different computation speeds. We still sum all the database times from the different machines because this number still corresponds to the total amount time spent by users waiting for a response from the database server. We just need to consider that moving a thread of computation from one machine to another may result in changes to the total database time. For example, moving a CPU heavy computation from a slow CPU to a fast CPU is going to reduce database time if the two machines are not CPU bound both before and after the move.

5.2 Background Activity

Background activity is an important component in an Oracle server. For example, data blocks do not need to be written to disk even after a transaction commits. It is enough that the appropriate log records are written to disk. Therefore, most write I/O to tables and indexes is done as a background activity; no user waits for these writes to complete.

We regard background activity as a set of threads of computation. The time spent in these threads is not counted in database time since no user waits for these threads. However, statistics about resource usage by the background must be maintained. When ADDM detects that a resource is used too much, a possible recommendation is to reduce the background activity that uses the resource. For example, if the I/O sub-system is used so heavily that I/O response time is both very slow and responsible for most of the database time, we might consider reducing the frequency of checkpoints.

5.3 Parallel Computation

The first problem we have with parallel computation is how to measure database time. Parallel computation implies that a single user connection is responsible for multiple threads of computation, which may use multiple CPUs and wait for multiple events or resources. Since database time is a measurement of throughput, we must consider all time spent by all the threads of computation as part of database time. The only exception is when one thread is waiting for another thread that belongs to the same user connection. In that case we consider the wait time as idle and we do not add it to database time. The reason for this exception is that we try to gauge what the database time would be if the computation were serialized while maintaining the same execution plan. In that case, all CPU usage and waits for external resources are still parts of the computation. However, internal wait between threads disappears from the computation.

Parallel computations are used when we need to reduce the response time of a request. As is always the case with parallel algorithms throughput is sacrificed to get better response time. In many cases, the response time of a specific user request is more important than the total system throughput. ADDM advises the database administrator how much extra database time is spent on parallel computations compared to serial computation. In this case we must consult the optimizer to find the best serial execution plan and compare it to the parallel execution plan that was actually used. This is a simulation since we do not execute the serial plan and do not know the exact cost in terms of database time.

5.4 Summary

We have seen how distributed systems, background activity and parallel queries can complicate both the notion of database time and the goal of an ADDM analysis. These are not the only complications we encountered and solved when implementing ADDM in Oracle 10g. However, a complete list is outside the scope of this paper. The general principle remains the same: find a common currency to measure database throughput, find ways to increase that throughput and improve the performance of the database as users experience it.

6. ADDM's Role in Self-Managing Databases

ADDM is a central part of the manageability framework for self-managing a database that was developed in Oracle 10g (see [ORM10]). This framework enables a comprehensive tuning solution by providing the necessary components. ADDM is a key component that serves as the central intelligence for various tuning activities in the system, like SQL Tuning [ORQ10] or Space Fragmentation Analysis [ORS10], since it takes a holistic view of the database system and identifies the top issues affecting overall throughput. ADDM itself relies on the framework to a large extent for ready access to the performance measurements it needs.

The manageability solution in Oracle 10g is centered around the three phases of the self-managing loop: *Observe*, *Diagnose*, and *Resolve*. Each component provided by the framework plays a key role in one or more of these phases. These phases refer to a particular activity (e.g.: a SQL tuning cycle), and there could be

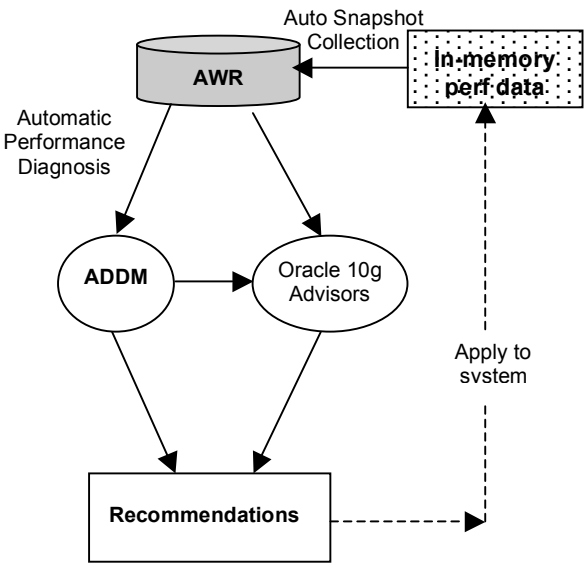


Figure 5: ADDM and the Self-Managing Database framework

many such activities occurring concurrently each in different phases. Figure 5 and the subsequent sections below illustrate the relationship between the main components and how they interact with ADDM.

6.1 Observe Phase

This phase is automatic and continuous in Oracle 10g and provides the data needed for analysis and feedback as a result of the actions taken as part of the analysis. To enable accurate system performance monitoring and tuning it is imperative that the system under consideration exposes relevant measurements and statistics. The manageability framework allows for instrumentation of the code to obtain precise timing information, and provides a lightweight comprehensive data collection mechanism to store these measurements for further online or offline analysis.

The main component of this phase is the "Automatic Workload Repository" (AWR). It is a persistent store of performance data for Oracle10g. The database captures system and performance data from in-memory views every hour and stores it in AWR. Each collection is referred to as a snapshot. Any pair of snapshots determines an analysis period that can be used for an ADDM analysis. The AWR is self-managing; it accepts policies for data retention and proactively purges data should it encounter space pressure.

The snapshot mechanism along with the comprehensive data it collects solves the "first occurrence analysis" requirement. ADDM runs automatically each time a snapshot is taken, the period of analysis being defined by the two most recent consecutive snapshots.

6.2 Diagnose Phase

The activities in this phase refer to the analysis of various parts of the database system using the data in AWR or in the in-memory views. Oracle 10g introduces a set of advisors, ADDM being one of them, for analyzing and optimizing the performance of its respective sub-components. Of particular note is the fact that all the advisors provide an impact in terms of Database Time for the problems they diagnose. This allows for easy comparison across advisors' results.

ADDM consults these advisors during its analysis depending on the amount of time and resources needed by these advisors. If an advisor could potentially take a substantial amount of time for its analysis, ADDM generates a recommendation to invoke that specific advisor instead. The user can then schedule this task when appropriate.

The set of advisors that ADDM can invoke include

- *SQL Tuning Advisor*: tunes SQL statements by improving the execution plan, by performing cost based access path analysis and SQL structure analysis.
- *Segment Advisor*: analyses space wastage by objects due to internal and external fragmentation.
- *Memory Advisors*: continuously monitor the database instance and auto-tune the memory utilization between the various memory pools.

6.3 Resolve Phase

The various advisors, after having performed their analysis, provide as output a set of recommendations that can be implemented or applied to the database. Each recommendation is accompanied by a benefit, in database time, that the workload would experience should the recommendation be applied. The recommendations may be automatically applied by the database (e.g., the memory resizing by the memory advisors) or it may be initiated manually. This is the Resolve phase.

Applying recommendations to the system close an iteration of that particular tuning loop. The influence of the recommendations on the workload will then be observed in future performance measurements. Further tuning loops may be initiated until the desired level of performance is attained.

7. Experiments and Results

It is not an easy task to test and quantify the results of ADDM. The value of ADDM is in helping DBAs manage the performance of their databases. Checking if ADDM meets the task, we need to survey many customers that already adapted Oracle 10g in a production or test environment. It is too early after the release of Oracle 10g to perform such a survey. However, we provide three examples of real ADDM usage in Sections 7.1 to 7.3.

We used various scenarios to gauge the effectiveness of ADDM.

Blind tests were performed in the initial testing phase; performance tuning experts were asked to analyse and make recommendations on running systems. In a majority of cases ADDM produced comparable results and in some cases ADDM's recommendations produced greater benefit than the experts' recommendations.

As part of the testing performed internal to Oracle Server Technologies group we have a number of tests in which we introduced known 'common faults' and application inefficiencies. When ADDM analysis was performed on the data captured during these tests ADDM correctly identified the fault in all cases.

High load stress testing is an integral part of the testing of the database product at Oracle and the effect of running all of the Oracle 10g manageability features was measured. With both typical customer workloads and highly tuned industry standard benchmarks the reduction in throughput from enabling all of the manageability features was approximately 3%. AWR and ADDM were two of the features enabled. ADDM runs performed after each AWR snapshot were available in a timely manner, typically under ten seconds.

Although Oracle 10g was declared production in January 2004 a number of internal production systems were running the software for several months before this date. ADDM has identified and correctly diagnosed a number of performance issues in this time on these systems.

7.1 Experience of Qualcomm

Qualcomm Centauri Application was upgraded from Oracle version 8.1.7.4 to 10g RAC. While testing the upgrade they found significant performance problems and testers reported poor response times. The DBA looked at the ADDM recommendations which highlighted a SQL (update statement) that was causing over 90% of the system load. They then ran SQL Tuning Advisor on the statement and it recommended the creation of an index. It was later found that the recommended index should have been in place. The index was missing because a patch to the application was applied to the production system but not to the upgraded test system. Identifying the index made problem diagnosis easy.

7.2 Experience in Oracle's Bug Database

The Oracle Bug database is used daily by many thousands of users and was one of the first production systems to move to Oracle 10g. The system runs on an 8 CPU PA-RISC HP machine. After upgrading to Oracle 10g users experienced poor performance. ADDM reported that users were spending a large proportion of their time waiting for free buffer waits and recommended examining the I/O subsystem for write performance (this type of problem happens when the buffer cache is filled with dirty buffers and faster I/O should solve the problem). When the System Administrator looked at the I/O subsystem he found that the asynchronous I/O was incorrectly configured causing asynchronous I/O requests to fail and then be performed synchronously leading to extremely slow IO times.

7.3 Experience of Oracle Applications QA Testing

An Applications Development DBA reported that a user said that the system was slow. Unfortunately there was no timescale or details given. Investigation was made harder by the fact that the users of the system and the DBAs investigating the slowdown where on opposite sides of the world, 12 time zones apart.

Looking at ADDM reports there were a couple of one hour periods in which the time spent in the database was significantly higher. Both of these ADDM reports showed that most of the time was spent in parsing and that the parsing was caused by the application generating large numbers of SQL statements containing literal string values. (This problem is Oracle's equivalent of using many similar SQL statements instead of a stored procedure. The cost of such configurations is time spent in parsing, optimizing and compiling the SQL statements-in Oracle's terminology it is called "hard parse"). The recommendation from ADDM was to modify the application to use bind variables rather than literals or to change a database configuration parameter to automatically convert the literals into binds. When application development was approached about the literal usage it was discovered that this QA system was running a 'known bad' internal build of the application and upgrading to the correct build removed the issue.

8. Conclusion

In this paper we described how Oracle 10g offers a comprehensive tuning solution by providing automatic performance diagnosis and tuning via ADDM. This addresses the ever-increasing performance tuning challenges currently faced by database administrators.

We defined a new measure called Database Time that allows for comparison of the performance impact of various database components with each other. We also described what types of performance measurements are needed for accurate performance diagnosis, as well as how we obtain them in a manner that has marginal impact on the system. This solution also obviates the need to reproduce a problem in order to diagnose it.

ADDM incorporates a holistic view of the system, and by using database time in conjunction with the two-dimensional DBTime-graph it is able to quickly isolate the root causes of performance bottlenecks affecting the throughput of the system. ADDM provides specific actionable recommendations with an estimate of the benefit for alleviating performance problems.

The results of running ADDM on a variety of workloads and production systems within Oracle demonstrates the benefits and practicality of our approach for throughput tuning.

Acknowledgements

The authors would like to thank all the members of the Server Manageability team at Oracle for their valuable feedback in all stages of this project. We would like to thank Connie Green for providing helpful comments on an earlier version of this paper.

References

[BE03] D. G. Benoit. Automatic Diagnosis of Performance Problems in Database Management Systems, PhD Thesis, Queen's University, Canada, 2003.

[BR94] K. P. Brown, M. Mehta, M.J. Carey, M. Livny: Towards Automated Performance Tuning for Complex Workloads. 20th International Conference on Very Large Data Bases, Santiago, Chile, 1994.

[CH97] S. Chaudhuri, V. Narasayya: An Efficient, Cost-driven Index Tuning Wizard for Microsoft SQL Server, 23rd International Conference on Very Large Data Bases, Athens, Greece, 1997.

[CH00] S. Chaudhuri and G. Weikum: Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System. 26th International Conference on Very Large Data Bases, Cairo, Egypt, 2000.

[DA02] B. Dageville, M. Zait: SQL Memory Management in Oracle9i, 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002.

[EL03] S. Elnaffar, W. Powley, D. Benoit, P. Martin: Today's DBMSs: How Autonomic Are They? Proceedings of the First IEEE International Autonomic Systems Workshop, Prague, DEXA 2003.

[GA96] Gartner Group: Total Cost of Ownership: The Impact of System Management Tools, 1996.

[HE97] J. L. Hellerstein. Automated Tuning Systems: Beyond Decision Support. In the proceedings on Computer Measurement Group, 1997

[HU02] Hurwitz Group: Achieving Faster Time-to-Benefit and Reduced TCO with Oracle Certified Configurations, March 2002.

[IBG8] IBM Corporation: DB2 Universal Database Version 8 Guide to GUI Tools for Administration and Development, IBM Corporation, 2003.

[IBP8] IBM Corporation: DB2 Universal Database Version 8 Administration Guide: Performance, IBM Corporation, 2003.

[MSPD] Microsoft Corporation: RDBMS Performance Tuning Guide for Data Warehousing, Chapter 20, SQL Server 2000 Resource Kit.

[ORM9] Oracle Corporation: Oracle 9i Database Manageability, Oracle White Paper, http://www.oracle.com/technology/products/manageability/database/pdf/Oracle9iManageabilityBWP.pdf

[ORM10] Oracle Corporation: Oracle Database 10g: The Self-Managing Database, Oracle White Paper, http://www.oracle.com/technology/products/manageability/database/pdf/twp03/TWP_manage_self_managing_database.pdf

[ORP8] Oracle Corporation: Oracle 8i Designing and Tuning for Performance Release 2, Oracle 8i Documentation, http://tahiti.oracle.com

[ORP9] Oracle Corporation: Oracle 9i Database Performance Tuning Guide and Reference, Oracle 9i Documentation, http://tahiti.oracle.com

[ORQ9] Oracle Corporation: Query Optimization in Oracle 9i, Oracle White Paper, http://www.oracle.com/technology/products/bi/pdf/o9i_optimization_twp.pdf

[ORQ10] Oracle Corporation: The Self-Managing Database: Guided Application and SQL Tuning, Oracle White Paper, http://www.oracle.com/technology/products/manageability/database/pdf/twp03/TWP_manage_automatic_SQL_tuning.pdf

[ORS10] Oracle Corporation: The Self-Managing Database: Proactive Space and Schema Object Management, Oracle Presentation, http://www.oracle.com/technology/products/manageability/database/pdf/ow03p/40170p.pdf

[ORSP] Oracle Corporation: Diagnosing Performance Bottlenecks using Statspack and The Oracle Performance Method, Oracle White Paper, http://www.oracle.com/technology/deploy/performance/pdf/statspack_opm4.pdf

[ORSP2] Oracle Corporation: Diagnosing Performance Using Statspack, Oracle White Paper, http://www.oracle.com/technology/deploy/performance/pdf/statspack.pdf

[WE94] G. Weikum, C. Hasse, A. Mönkeberg, P. Zabback: The COMFORT Automatic Tuning Project, Information Systems, Vol. 19, No. 5, pp. 381-432, 1994.

[WE02] G. Weikum, A. Mönkeberg, C. Hasse, P. Zabback: Selftuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering, 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002.