3
Performance Improvement Methods

This chapter discusses Oracle Database improvement methods and contains the following sections:

The Oracle Performance Improvement Method

Oracle performance methodology helps you to identify performance problems in an Oracle database. This involves identifying bottlenecks and fixing them. It is recommended that changes be made to a system only after you have confirmed that there is a bottleneck.

Performance improvement, by its nature, is iterative. For this reason, removing the first bottleneck might not lead to performance improvement immediately, because another bottleneck might be revealed. Also, in some cases, if serialization points move to a more inefficient sharing mechanism, then performance could degrade. With experience, and by following a rigorous method of bottleneck elimination, applications can be debugged and made scalable.

Performance problems generally result from either a lack of throughput, unacceptable user/job response time, or both. The problem might be localized between application modules, or it might be for the entire system.

Before looking at any database or operating system statistics, it is crucial to get feedback from the most important components of the system: the users of the system and the people ultimately paying for the application. Typical user feedback includes statements like the following:

- "The online performance is so bad that it prevents my staff from doing their jobs."
- "The billing run takes too long."
- "When I experience high amounts of Web traffic, the response time becomes unacceptable, and I am losing customers."
- "I am currently performing 5000 trades a day, and the system is maxed out. Next month, we roll out to all our users, and the number of trades is expected to quadruple."

From candid feedback, it is easy to set critical success factors for any performance work. Determining the performance targets and the performance engineer's exit criteria make managing the performance process much simpler and more successful at all levels. These critical success factors are better defined in terms of real business goals rather than system statistics.

Some real business goals for these typical user statements might be:

- "The billing run must process 1,000,000 accounts in a three-hour window."
- "At a peak period on a Web site, the response time must not exceed five seconds for a page refresh."
- "The system must be able to process 25,000 trades in an eight-hour window."

The ultimate measure of success is the user's perception of system performance. The performance engineer's role is to eliminate any bottlenecks that degrade performance. These bottlenecks could be caused by inefficient use of limited shared resources or by abuse of shared resources, causing serialization. Because all shared resources are limited, the goal of a performance engineer is to maximize the number of business operations with efficient use of shared resources. At a very high level, the entire database server can be seen as a shared resource. Conversely, at a low level, a single CPU or disk can be seen as shared resources.

You can apply the Oracle performance improvement method until performance goals are met or deemed impossible. This process is highly iterative. Inevitably, some investigations may have little or no impact on database performance. Time and experience are necessary to develop the skills to accurately and quickly pinpoint critical bottlenecks. However, prior experience can sometimes work against the experienced engineer who neglects to use the data and statistics available. This type of behavior encourages database tuning by myth and folklore. This is a very risky, expensive, and unlikely to succeed method of database tuning.

The Automatic Database Diagnostic Monitor (ADDM) implements parts of the performance improvement method and analyzes statistics to provide automatic diagnosis of major performance issues. Using ADDM can significantly shorten the time required to improve the performance of a system.

Systems are so different and complex that hard and fast rules for performance analysis are impossible. In essence, the Oracle performance improvement method defines a way of working, but not a definitive set of rules. With bottleneck detection, the only rule is that there are no rules! The best performance engineers use the data provided and think laterally to determine performance problems.

Steps in the Oracle Performance Improvement Method

1. Perform the following initial standard checks:
    a. Get candid feedback from users. Determine the performance project's scope and subsequent performance goals, and performance goals for the future. This process is key in future capacity planning.
    b. Get a full set of operating system, database, and application statistics from the system when the performance is both good and bad. If these are not available, then get whatever is available. Missing statistics are analogous to missing evidence at a crime scene: They make detectives work harder and it is more time-consuming.
    c. Sanity-check the operating systems of all computers involved with user performance. By sanity-checking the operating system, you look for hardware or operating system resources that are fully utilized. List any over-used resources as symptoms for analysis later. In addition, check that all hardware shows no errors or diagnostics.
2. Check for the top ten most common mistakes with Oracle Database, and determine if any of these are likely to be the problem. List these as symptoms for later analysis. These are included because they represent the most likely problems. ADDM automatically detects and reports nine of these top ten issues.
3. Build a conceptual model of what is happening on the system using the symptoms as clues to understand what caused the performance problems. See "A Sample Decision Process for Performance Conceptual Modeling".
4. Propose a series of remedy actions and the anticipated behavior to the system, then apply them in the order that can benefit the application the most. ADDM produces recommendations each with an expected benefit. A golden rule in performance work is that you only change one thing at a time and then measure the differences. Unfortunately, system downtime requirements might prohibit such a rigorous investigation method. If

multiple changes are applied at the same time, then try to ensure that they are isolated so that the effects of each change can be independently validated.

5. Validate that the changes made have had the desired effect, and see if the user's perception of performance has improved. Otherwise, look for more bottlenecks, and continue refining the conceptual model until your understanding of the application becomes more accurate.
6. Repeat the last three steps until performance goals are met or become impossible due to other constraints.

This method identifies the biggest bottleneck and uses an objective approach to performance improvement. The focus is on making large performance improvements by increasing application efficiency and eliminating resource shortages and bottlenecks. In this process, it is anticipated that minimal (less than 10%) performance gains are made from instance tuning, and large gains (100% +) are made from isolating application inefficiencies.

A Sample Decision Process for Performance Conceptual Modeling

Conceptual modeling is almost deterministic. However, as you gain experience in performance tuning, you begin to appreciate that no real rules exist. A flexible heads-up approach is required to interpret statistics and make good decisions.

For a quick and easy approach to performance tuning, use ADDM. ADDM automatically monitors your Oracle system and provides recommendations for solving performance problems should problems occur. For example, suppose a DBA receives a call from a user complaining that the system is slow. The DBA simply examines the latest ADDM report to see which of the recommendations should be implemented to solve the problem.

The following steps illustrate how a performance engineer might look for bottlenecks without using automatic diagnostic features. These steps are only intended as a guideline for the manual process. With experience, performance engineers add to the steps involved. This analysis assumes that statistics for both the operating system and the database have been gathered.

1. Is the response time/batch run time acceptable for a single user on an empty or lightly loaded computer?

   If it is not acceptable, then the application is probably not coded or designed optimally, and it will never be acceptable in a multiple user situation when system resources are shared. In this case, get application internal statistics, and get SQL Trace and SQL plan information. Work with developers to investigate problems in data, index, transaction SQL design, and potential deferral of work to batch and background processing.
2. Is all the CPU being utilized?

   If the kernel utilization is over 40%, then investigate the operating system for network transfers, paging, swapping, or process thrashing. Continue to check CPU utilization in user space to verify if there are any non-database jobs consuming CPU on the system limiting the amount of shared CPU resources, such as backups, file transforms, print queues, and so on. After determining that the database is using most of the CPU, investigate the top SQL by CPU utilization. These statements form the basis of all future analysis. Check the SQL and the transactions submitting the SQL for optimal execution. Oracle Database provides CPU statistics in `V$SQL` and `V$SQLSTATS`.

> **See Also:**
>
> *Oracle Database Reference* for more information about `V$SQL` and `V$SQLSTATS`

If the application is optimal and no inefficiencies exist in the SQL execution, then consider rescheduling some work to off-peak hours or using a bigger computer.

3. At this point, the system performance is unsatisfactory, yet the CPU resources are not fully utilized.

   In this case, you have serialization and unscalable behavior within the server. Get the `WAIT_EVENTS` statistics from the server, and determine the biggest serialization point. If there are no serialization points, then the problem is most likely outside the database, and this should be the focus of investigation. Elimination of `WAIT_EVENTS` involves modifying application SQL and tuning database parameters. This process is very iterative and requires the ability to drill down on the `WAIT_EVENTS` systematically to eliminate serialization points.

Top Ten Mistakes Found in Oracle Systems

This section lists the most common mistakes found in Oracle databases. By following the Oracle performance improvement methodology, you should be able to avoid these mistakes altogether. If you find these mistakes in your system, then re-engineer the application where the performance effort is worthwhile.

1. Bad connection management

   The application connects and disconnects for each database interaction. This problem is common with stateless middleware in application servers. It has over two orders of magnitude impact on performance, and is totally unscalable.

2. Bad use of cursors and the shared pool

   Not using cursors results in repeated parses. If bind variables are not used, then there is hard parsing of all SQL statements. This has an order of magnitude impact in performance, and it is totally unscalable. Use cursors with bind variables that open the cursor and execute it many times. Be suspicious of applications generating dynamic SQL.

3. Bad SQL

   Bad SQL is SQL that uses more resources than appropriate for the application requirement. This can be a decision support systems (DSS) query that runs for more than 24 hours, or a query from an online application that takes more than a minute. You should investigate SQL that consumes significant system resources for potential improvement. ADDM identifies high load SQL. SQL Tuning Advisor can provide recommendations for improvement.

4. Use of nonstandard initialization parameters

   These might have been implemented based on poor advice or incorrect assumptions. Most databases provide acceptable performance using only the set of basic parameters. In particular, parameters associated with `SPIN_COUNT` on latches and undocumented optimizer features can cause a great deal of problems that can require considerable investigation.

   Likewise, optimizer parameters set in the initialization parameter file can override proven optimal execution plans. For these reasons, schemas, schema statistics, and optimizer settings should be managed as a group to ensure consistency of performance.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about initialization parameters and database creation
> - *Oracle Database Reference* for details on initialization parameters

5. Getting database I/O wrong

   Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly, because they configure disks by disk space and not I/O bandwidth.

6. Online redo log setup problems

   Many sites run with too few online redo log files and files that are too small. Small redo log files cause system checkpoints to continuously put a high load on the buffer cache and I/O system. If too few redo log files exist, then the archive cannot keep up, and the database must wait for the archiver to catch up.

7. Serialization of data blocks in the buffer cache due to lack of free lists, free list groups, transaction slots (`INITRANS`), or shortage of rollback segments.

   This is particularly common on `INSERT`-heavy applications, in applications that have raised the block size above 8K, or in applications with large numbers of active users and few rollback segments. Use automatic segment-space management (ASSM) and automatic undo management to solve this problem.

8. Long full table scans

   Long full table scans for high-volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O intensive and unscalable.

9. High amounts of recursive (`SYS`) SQL

   Large amounts of recursive SQL executed by `SYS` could indicate space management activities, such as extent allocations, taking place. This is unscalable and impacts user response time. Use locally managed tablespaces to reduce recursive SQL due to extent allocation. Recursive SQL executed under another user ID is probably SQL and PL/SQL, and this is not a problem.

10. Deployment and migration errors

    In many cases, an application uses too many resources because the schema owning the tables has not been successfully migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. These errors can lead to sub-optimal execution plans and poor interactive user performance. When migrating applications of known performance, export the schema statistics to maintain plan stability using the `DBMS_STATS` package.

    Although these errors are not directly detected by ADDM, ADDM highlights the resulting high load SQL.

Emergency Performance Methods

This section provides techniques for dealing with performance emergencies. You presumably have a methodology for establishing and improving application performance. However, in an

emergency situation, a component of the system has changed to transform it from a reliable, predictable system to one that is unpredictable and not satisfying user requests.

In this case, the performance engineer must rapidly determine what has changed and take appropriate actions to resume normal service as quickly as possible. In many cases, it is necessary to take immediate action, and a rigorous performance improvement project is unrealistic.

After addressing the immediate performance problem, the performance engineer must collect sufficient debugging information either to get better clarity on the performance problem or to at least ensure that it does not happen again.

The method for debugging emergency performance problems is the same as the method described in the performance improvement method earlier in this book. However, shortcuts are taken in various stages because of the timely nature of the problem. Keeping detailed notes and records of facts found as the debugging process progresses is essential for later analysis and justification of any remedial actions. This is analogous to a doctor keeping good patient notes for future reference.

Steps in the Emergency Performance Method

The Emergency Performance Method is as follows:

1. Survey the performance problem and collect the symptoms of the performance problem. This process should include the following:
   - User feedback on how the system is underperforming. Is the problem throughput or response time?
   - Ask the question, "What has changed since we last had good performance?" This answer can give clues to the problem. However, getting unbiased answers in an escalated situation can be difficult. Try to locate some reference points, such as collected statistics or log files, that were taken before and after the problem.
   - Use automatic tuning features to diagnose and monitor the problem. In addition, you can use Oracle Enterprise Manager Cloud Control (Cloud Control) performance features to identify top SQL and sessions.
2. Sanity-check the hardware utilization of all components of the application system. Check where the highest CPU utilization is, and check the disk, memory usage, and network performance on all the system components. This quick process identifies which tier is causing the problem. If the problem is in the application, then shift analysis to application debugging. Otherwise, move on to database server analysis.
3. Determine if the database server is constrained on CPU or if it is spending time waiting on wait events. If the database server is CPU-constrained, then investigate the following:
   - Sessions that are consuming large amounts of CPU at the operating system level and database; check `V$SESS_TIME_MODEL` for database CPU usage
   - Sessions or statements that perform many buffer gets at the database level; check `V$SESSTAT` and `V$SQLSTATS`
   - Execution plan changes causing sub-optimal SQL execution; these can be difficult to locate
   - Incorrect setting of initialization parameters
   - Algorithmic issues caused by code changes or upgrades of all components

   If the database sessions are waiting on events, then follow the wait events listed in `V$SESSION_WAIT` to determine what is causing serialization. The

V$ACTIVE_SESSION_HISTORY view contains a sampled history of session activity which you can use to perform diagnosis even after an incident has ended and the system has returned to normal operation. In cases of massive contention for the library cache, it might not be possible to logon or submit SQL to the database. In this case, use historical data to determine why there is suddenly contention on this latch. If most waits are for I/O, then examine V$ACTIVE_SESSION_HISTORY to determine the SQL being run by the sessions that are performing all of the inputs and outputs.

4. Apply emergency action to stabilize the system. This could involve actions that take parts of the application off-line or restrict the workload that can be applied to the system. It could also involve a system restart or the termination of job in process. These naturally have service level implications.
5. Validate that the system is stable. Having made changes and restrictions to the system, validate that the system is now stable, and collect a reference set of statistics for the database. Now follow the rigorous performance method described earlier in this book to bring back all functionality and users to the system. This process may require significant application re-engineering before it is complete.