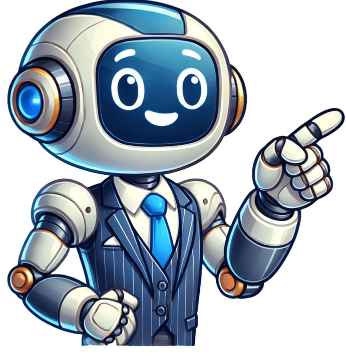
Continue



Have you noticed your database running slower than usual? Then you might need a touch of performance tuning. Performance tuning in Oracle databases eliminates bottlenecks that make applications unresponsive and increases the load your database can handle. What is Oracle Performance Tuning? Oracle performance tuning is an essential practice for database administrators (DBAs) and developers aiming to optimize the performance of Oracle databases. As the backbone of many enterprise applications, Oracle databases handle vast amounts of data and complex transactions, making performance a critical factor in ensuring system efficiency, reliability, and scalability. Poor performance can lead to slow query response times, resource contention, and system outages, ultimately affecting business operations. This guide serves as an introduction to Oracle performance tuning, providing practical insights into key areas of optimization, including SQL query tuning, memory management, disk I/O optimization, and proper indexing strategies. It will cover the most common performance bottlenecks and offer proven techniques for identifying, diagnosing, and resolving issues that may hinder database performance. Effective performance tuning requires a combination of deep technical knowledge, an understanding of the specific workload requirements, and a disciplined approach to continuous monitoring. Through this guide, you will learn how to assess your Oracle database's current state, make targeted improvements, and apply best practices for maintaining peak performance over time. This guide will equip you with the essential tools and strategies to optimize your Oracle database, ensuring it operates efficiently and supports your mission-critical applications with optimal speed and reliability. Monitoring Performance in Oracle Databases: Response Time and Throughput When performance tuning an Oracle database there are two metrics that are useful to measure: Response time – How long the database takes to complete a request. System throughput – The number of processes completed in a period of time. High response time means that an application is providing a slow user experience. On the other hand, low system throughput means that the database only has the resources to manage a small number of tasks in a short time period. An administrator has to be able to know how they are trying to improve performance before tuning. How you optimize an Oracle database comes down to your goals and the type of applications you are using. Many goals like having a fast response time or a high throughput are contradictory. Tuning for fast response times may speed up individual queries made by users but sacrifice other tasks in the workload. In contrast, achieving a high throughput would aim to optimize the performance of the entire workload to support a larger output of transactions per second (but not necessarily speed up individual queries). The type of application you're using makes all the difference. If you're using an online transaction process (OLTP) application then you would use throughput to measure performance. This is because of the high volume of transactions the application needs to manage. However, if you were using a decision support system (DSS) with users running queries on everything from a handful of records to thousands of records, then you would measure performance by response time (unless you were supporting lots of users running concurrent queries)! The Two Types of Tuning: Proactive Monitoring and Bottleneck Elimination Now that you know what performance tuning is, it's important to look at the two main models of tuning: Proactive Monitoring Bottleneck Elimination Database administrators use these two models to manage performance issues and keep applications functioning at a high level. Proactive Monitoring Proactive monitoring is the process of monitoring a database to discover and address performance issues early rather than simply reacting when there is a problem. With proactive monitoring, administrators will periodically review databases to identify the signs of performance degradation. The idea behind proactive monitoring is to catch issues and inefficiencies before they develop into greater problems further down the line. Some common issues database administrators look out for include: Database wait events – A high number of events can negatively affect database performance. Finding obstructive sessions and killing them can prevent performance degradation. Load average – Monitoring the load average of a server will tell you if server resources are functioning as normal. A high load average can result in slow database performance. Database sessions – Monitoring the number of active sessions can stop you from reaching the maximum (which will prevent you from being able to open new sessions). However, monitoring proactively does carry some risk. Any changes an administrator makes can result in a decrease in performance for the database. Administrators can mitigate the risks by being cautious before making new changes. Bottleneck Elimination Bottlenecks are one of the most common causes of poor performance. Bottlenecks block requests from reaching the destination and increase the response time of applications. Bottlenecks can be caused by a range of factors from badly coded SQL statements and high resource usage. Bottleneck elimination is more of a reactive process than proactive monitoring. An administrator identifies a bottleneck and then finds a way to fix it. Fixing a bottleneck is a complex process and depends on what the root cause is (and whether it is internal or external). Recoding SQL statements is one solution for fixing internal bottlenecks, which should be addressed first. Once internal bottlenecks have been resolved the administrator can start to look at external factors like CPU and storage performance that could be causing the problem. An administrator can choose between making changes to the application (or how it is used), Oracle, or the hardware configuration of the host. How to Performance Tune Performance tuning an Oracle database is a very complex subject because there are so many different factors that can affect database performance. To keep things simple, we're going to look at some basic ways you can optimize performance. 1. Identify High-Cost Queries The first step to tuning SQL code is to identify high-cost queries that consume excessive resources. Rather than optimizing every line of code it is more efficient to focus on the most widely-used SQL statements and have the largest database / I/O footprint. One easy way to identify high-cost queries is to use Oracle database monitoring tools (we look at some of these platforms in more detail further down below). One useful tool is Oracle SQL Analyze, which can identify resource-intensive SQL statements. Tuning these statements will give you the greatest return on your time investment. 2. Minimize the workload (Use Indexes!) You can make the same query in many different ways so it is advantageous to write code that minimizes the workload as much as possible. If you only need to display a small number of records then you should add an index to your query. A full table scan takes up more resources and I/O. To eliminate the stress of sustaining a large workload you can use indexes to access small sets of rows rather than processing the entire database at once. Use indexes in those scenarios where a column is regularly queried. 3. Use Stateful Connections with Applications Sometimes the cause of poor performance doesn't come from code but because the connection keeps dropping between the application and the database. If your application isn't configured correctly then it could form a connect to the database to access a table and then drop the connection once it has the information it needs. Dropping the connection after accessing the table is terrible for performance. Instead, try to keep a stateful connection so that the application stays connected to the database at all times. Maintaining the connection will stop system resources from being wasted each time the application interacts with the database. 4. Collect and Store Optimizer Statistics Optimizer statistics are data that describe a database and its objects. These statistics are used by the database to choose the best execution plan for SQL statements. Regularly collecting and storing optimizer statistics on database objects is essential for maintaining efficiency. Collecting optimizer statistics makes sure that the database has accurate information on table contents. If the data is inaccurate then the database can choose a poor execution plan, which will affect the end-user experience. Oracle databases can automatically collect optimizer statistics or you can do so manually with the DBMS_STATS package. Using a software agent to monitor SQL query performance is the most effective way to manage query performance. Database performance monitoring tools can help to identify poorly performing SQL code and enable you to pinpoint the root cause of performance issues. The user can view query performance on a dashboard so they don't need to search for information manually. SolarWinds Database Performance Analyzer (FREE TRIAL) SolarWinds Database Performance Analyzer is a tool that can monitor the performance of SQL queries in real-time. You can view a table of the Top SQL Statements in your database to see the highest impact SQL statements. Viewing the top statements allows you to focus your remediation efforts on those statements that have the greatest impact on performance. Why do we recommend it? The SolarWinds Database Performance Analyzer provides cloud-based tracking of activities and issues on Oracle databases on your own servers and on cloud platforms. The system uses AI to predict when performance issues are gathering and issue warnings for action before problems become noticeable to users. Who is it recommended for? This SaaS package is able to manage and monitor a list of DBMS's, not just Oracle. It can also watch over database instances from SQL Server, MySQL, MariaDB, PostgreSQL, and MongoDB as well as other relational and NoSQL systems. As it only monitors databases and their hosts, the tool is aimed at large businesses. Pros: Highly intuitive DB management system tailored for medium to large size database implementations Monitors in real-time, offering a number of alert and notification options that can integrate into popular helpdesk solutions Threshold monitoring helps keep teams proactive, and fix issues before they impact performance Dashboards are highly customizable and can be tailored to individuals or teams Built-in query analysis helps DBAs build more efficient queries Leverages machine learning to identify performance bottlenecks Cons: Would prefer a longer trial period There is also a blocking analysis feature, which allows you to view the blocking hierarchy of the database and to view the total wait caused. Being able to see those queries that are obscuring the routes of other queries tells you where to make changes to improve performance. You can download a free trial. SolarWinds Database Performance Analyzer Start 14-day FREE Trial dbForge Studio for Oracle (Oracle SQL Profiler) dbForge Studio for Oracle is an integrated development environment (IDE) that comes with Oracle SQL Profiler. With the profiler, you can identify queries with the highest duration. You can also view query execution plans and sessions statistics for additional information. Any changes you make will be saved so you can revert to earlier versions if you make a mistake. Why do we recommend it? dbForge Studio for Oracle is an on-device package that presents a view of database objects and allows queries to be tested and supports the development of code to run on your Oracle instances. This system can also be used to test database structures rather than for live monitoring. In the Session Statistics tab, you can compare the results of new queries against older versions. Differences are highlighted in red and green so you can easily tell the performance impact. Session stats you can monitor include, CPU used by this session and sorts (rows). The Session Statistics tab allows you to make sure that your changes are actually improving performance! Who is it recommended for? This tool runs on Windows, macOS, and Linux. It isn't a networked package but it can access databases across a network. This gives the tool limited usage within a business because you have to install the software on each user's workstation. The system is useful for DBAs and developers. It doesn't provide database monitoring. Pros: Offers additional features and functionality beyond Management Studio without unnecessary tools and options Excellent overview into the health of multiple SQL server databases Can rewrite queries and profile data from directly inside the tool Cons: Only runs on Windows Requires Microsoft SQL Server Management Studio Tune with a Goal in Mind! Having clear expectations with a specific goal in mind is critical for maintaining your database and tuning the system. Performance tuning is easier when you know what you're trying to achieve. For example, if you want to minimize the response time of an application so that it completes queries in 1-3 seconds then you can take action to tune for that scenario. That means diagnosing bottlenecks and performance issues that slow response time down. Being able to refer back to goals will also help when you're using a database analyzer. You'll be able to monitor disk, CPU, and network usage to identify how performance is affected. The better you understand your goals, the better you know how to tune your resources. Oracle performance tuning FAQs Indexing improves data retrieval speed. However, don't over-index because these indexes could conflict and actually slow down results being returned. Partitioning tables improves data access speed because it reduces the number of records that need to be scanned in order to return the results for a query. Partitioning takes all indexes currently on the table and duplicates them automatically so that they are automatically applied to each partition. SQL performance tuning is the process of ordering joins in an SQL query so that they extract records at maximum speed. There are a number of methods that can be used to improve SQL queries, such as removing outer joins and ordering lines in the WHERE clause so that as many records as possible are filtered out from the first table accessed by the query, thus reducing the amount of sorting that needs to be undertaken in the other tables accessed by the query. A bottleneck is a point in the process of access to or response from a database that has limited capacity and therefore slows disown access to results because only a limited number of queries can be served. This is different from resource locks, which occur when one process holds a resource and will not release it until another resource is available. I will share with you Step by Step Performance Tuning and SQL Tuning Tutorial in the Oracle Database in this article.Performance Tuning and SQL Tuning Tutorial in the OracleIf you don't know Performance Tuning and SQL Tuning in the Oracle database and want to learn, you are at the right adress.You can learn how to improve Oracle database Performance step by step with following steps. I have started Performance Tuning and SQL Tuning Tutorials from beginning, so read each article in order. Oracle Performance TuningLesson 1- What is Oracle Database Architecture.Oracle Database Architecture -1 Controlfile, Datafile, SGA and PGAOracle Database Architecture -2 Buffer Cache, Redolog Buffer, Onlinelog and ArchivelogOracle Database Architecture -3 SMON, PMON, DBWR,CKPT and LGWR Processes Lesson 2- SQL Statement Processing in Oracle Database SQL Statement Processing in Oracle DatabaseLesson 3- Introduction to Performance Management and TuningIntroduction to Performance Management and Tuning in Oracle | Oracle Database Performance Tuning Tutorial -1Lesson 4- Performance Tuning Features and Tools || What is the AWR Report and How to Analyze an AWR ReportPerformance Tuning Features and Tools in Oracle | Oracle Database Performance Tuning Tutorial -2What is the Oracle AWR (Automatic Workload Repository) ReportHow to Generate AWR (Automatic Workload Repository) Report via SQL*Plus, Enterprise Manager and Toad in OracleHow to Read or Analyze an AWR (Automatic Workload Repository) Report in Oracle -1How to Read or Analyze an AWR (Automatic Workload Repository) Report in Oracle -2How to Read or Analyze an AWR (Automatic Workload Repository) Report in Oracle -3How to Read or Analyze an AWR Report in Oracle -4 | TOP SQLOracle Wait Events and Their Solutions in Oracle DatabaseLesson 5- What is the Oracle Optimizer and How Cost-Based Optimizer Works ?What is the Oracle Optimizer and How Cost-Based Optimizer Works ? | Oracle Database Performance Tuning Tutorial -3 Lesson 6- What is the Execution Plan and How to View Execution PlanWhat is the Execution Plan and How to View Execution Plan in Oracle | Oracle Database Performance Tuning Tutorial -4Lesson 7- How to Interpret an Execution PlanHow to Read or Interpret an Execution Plan in Oracle | Oracle Database Performance Tuning Tutorial -5Lesson 8- What is the SQL Trace and How to Use SQL Trace , Trcsess and TKPROFHow to Use SQL Trace , Trcsess and TKPROF | Oracle Database Performance Tuning Tutorial -6 Lesson 9- What is the Index and Optimizer Access Paths (Table Scan and Index Scan)What is an Index (Advantages and Disadvantages) in Oracle DatabaseIndex Types and Index Create in Oracle DatabaseOptimizer Access Paths (Table and Index Scan) in Oracle | Oracle Database Performance Tuning Tutorial -7Lesson 10- Optimizer Access Paths & Join Operators (Hash Join, Nested Loop)Optimizer Access Paths & Join Operators (Hash Join, Nested Loop) in Oracle | Oracle Database Performance Tuning Tutorial -8Lesson 11- What is the SQL Tuning Set and How to Create the SQL Tuning SetsWhat is the SQL Tuning Set and How to Create the SQL Tuning Sets in Oracle | Oracle Database Performance Tuning Tutorial -9 Lesson 12- What is the SQL Tuning Advisor and How to Run SQL Tuning Advisor via Manually Enterprise Manager Cloud ControlWhat is the SQL Tuning Advisor and How to Run SQL Tuning Advisor Manually in Oracle | Oracle Database Performance Tuning Tutorial -10How to Run SQL Tuning Advisor via Enterprise Manager Cloud Control | Oracle Database Performance Tuning Tutorial -13Lesson 13- What is the SQL Access Advisor and How to Run SQL Access AdvisorWhat is the SQL Access Advisor and How to Run SQL Access Advisor | Oracle Database Performance Tuning Tutorial -11Lesson 14- What is the Automatic SQL Tuning and How to Automated SQL TuningWhat is the Automatic SQL Tuning and How to Automated SQL Tuning in Oracle | Oracle Database Performance Tuning Tutorial -12 Lesson 15- SQL Plan Management Using SQL Plan Baselines in Oracle | Oracle Database Performance Tuning Tutorial -14How to Create SQL Plan Baselines Using Enterprise Manager Cloud Control | Oracle Database Performance Tuning Tutorial -15Lesson 16- Optimizer Hints and How to Use Hints in OracleWhat is the Optimizer Hints and How to Use Hints in Oracle | Oracle Database Performance Tuning Tutorial -16Lesson 17- Automatic Diagnostic Repository Command Interpreter (ADRCI)Automatic Diagnostic Repository Command Interpreter (ADRCI)Lesson 18- SQL Performance Analyzer and SPA Report in OracleWhat is the SQL Performance Analyzer and SPA Report in Oracle | Oracle Database Performance Tuning Tutorial -17Lesson 19- Oracle SQLTXPLAIN (SQLT) Tips and TricksOracle SQLTXPLAIN (SQLT) Tips and Tricks | Oracle Database Performance Tuning Tutorial -18Lesson 20- SQL Tuning Tips and TricksSQL Tuning Tips and Tricks Tutorial in Oracle -1SQL Tuning Tips and Tricks Tutorial in Oracle -2SQL Tuning Tips and Tricks Tutorial in Oracle -3Lesson 21- Performance Tuning and Monitoring Scripts for Oracle DBAPerformance Tuning and SQL Monitoring Scripts | Oracle DBA Scripts All in One -4Find Blocking Sessions and Kill Locked Session Scripts & Locked Objects | Oracle DBA Scripts All in One -3Do you want to learn Oracle Database for Beginners, then Click and read the following articles.Oracle Database Tutorials for Beginners (Junior Oracle DBA) Ever wondered why Oracle Performance Tuning is so thrilling for budding Oracle developers and DBAs? The answer lies in the art of transforming a sluggish database into a finely-tuned powerhouse. At first, it might seem as complex as decoding a mad scientist's experiment. But fear not! This article is your guide, turning the daunting into doable with practical examples. Soon, tuning Oracle performance will become second nature to you. And the end result? Turning your Oracle into a speed powerhouse will make your users and your boss happy. You'll also level up your game to become an Oracle expert. So, let's dive into this challenging but fun journey. Here's the itinerary: Ready? Let's begin this exhilarating journey! If you plan to jump into the examples right away, PLEASE STOP. Understand this fundamental truth — it's not just about tools. It's about mastering the basics that lay the foundation for effective tuning. Performance tuning is the heartbeat of Oracle-powered systems. It's the key to unleashing the full potential of your Oracle database. If your app or website uses Oracle to store data, you need to make sure it's always fast. Or your users will get pissed off. Do you know that you only get 16 seconds tops before that happens? So, if you don't want to lose customers or something with the same heavy impact, make performance a priority. Know What Your Oracle License Covers Oracle's performance tuning tools are potent. But be aware—they might be behind a paywall. The Oracle Database Tuning Pack, for instance, requires an extra license. Consult Oracle support to unlock these powerful tools for fine-tuning. Understand Oracle Database Architecture This isn't just for DBAs. Database designers and developers must grasp Oracle's query processing lingo. To create high-performing databases and apps, you need to have a good database design. Then, dance with the optimizer and execution plans. Tune in to how Oracle databases move and groove. Know SQL and PL/SQL WellEven in a world of graphical tools, SQL newbies can get lost without understanding the syntax. To effectively tune queries, you must know the language. SQL and PL/SQL mastery is your tuning compass. Use Tools That are Simple and Increase ProductivityNo fancy tools? No problem. Dive into the V$ Performance views to spot and study sluggish queries. If you're eyeing a tool, make sure it's not just a pretty face—opt for productivity boosters. In this journey, we'll be wielding the power of dbForge Studio for Oracle. But before we continue our journey, let's have a short detour on some myths about database performance tuning. It's a common misconception that throwing more hardware at performance issues is a silver bullet. In reality, hardware upgrades don't address underlying inefficiencies. They are only temporary. Performance tuning is not reserved for giants; small and mid-sized databases benefit equally. Neglecting tuning based on the size of your database can lead to suboptimal performance even in compact systems. Contrary to belief, SQL tuning is a collective responsibility. Developers, database designers, and DBAs all play a role. Efficient SQL directly impacts performance, and a collaborative effort ensures a well-tuned database. Believing that indiscriminate indexing is the cure-all. While indexes boost performance, too many or improper indexes can introduce overhead. Database write performance degrades with unnecessary indexes. A strategic indexing approach is essential for optimal results. While Oracle has automatic indexing that could solve indexing problems, it's also not a cure-all. Seasoned administrators use a hybrid approach. The notion is that upgrading to the latest Oracle version magically fixes performance issues. Upgrades bring enhancements, but successful implementation demands meticulous planning and thorough testing. Relying solely on caching mechanisms as a performance magic pill is just a dream. True, caching helps. But well-optimized queries and database design remain pivotal for sustained high performance. Assuming that partitioning a table guarantees scalability. While it aids in managing large datasets, it's not a universal solution. Careful consideration of database architecture is crucial. Thinking that cloud databases automatically optimize performance. Even in the cloud, performance tuning is indispensable for unlocking the full potential of your database. And for your information, the samples we have later will use Oracle 19c in Oracle Cloud. You will see that there's something else you need to optimize. Now that you know these myths exist, avoid it like a plague. Instead, focus on how to add a turbo boost to your database. Let's discuss this in the next section. If you think you will see tips like "Avoid SELECT *" or "Avoid SELECT DISTINCT", these are not your usual tips. Those are not bad tips at all. But you'll find the tips below to be obvious as well. These are still basic compared to a ton of performance discussion in the Oracle documentation. But it all starts in the following tips. Let's begin. Normalize your Oracle database using the right data types and sizes Can you imagine getting a sumo wrestler? Their average weight is around 330 pounds. So, compared to an average American who weighs 197 pounds, sumo wrestlers are on the heavy side. And if your database is like that, your server will have a hard time running them. So, what is a sumo-wrestler-type Oracle database? It's the one that has tables with hundreds of columns. Even more, the columns are overweight too, like using CHAR(2000). And when you check it, you'll be surprised that the actual data is not even half of it. And the overall database design? It's only partially normalized. The effect? Fat columns consume more storage and RAM. In the end, queries are slower. So, what's a lean Oracle database? It's the one that uses the smallest possible data type for a column. Check out the Oracle data types and their storage requirements here. Make sure your design also follows normalization rules for a good relational database. Avoid chained rows If you've never heard of chained rows in Oracle, read this carefully. Chained rows can impact performance by increasing disk input/output. For starters, Oracle stores table rows in data blocks. An ideal scenario is fitting one row in a single data block. The default data block size is 8,192 bytes. But if your table design does not consider this, then a row might not fit in a single data block. When that happens, Oracle needs to put the other part of a row in another data block. And then, Oracle will "chain" them to connect one or more data blocks to store a single row. Chained rows happen in the following circumstances: Your table has more than 255 columns. When one of the table columns has a data type of LONG or LONG RAW. During the initial INSERT, the row fits a data block. But when an UPDATE occurs, the row length increases so that it won't fit a data block. This happens when one or more of the columns is VARCHAR. The bigger value during the UPDATE of the VARCHAR column(s) exceeds the size of the data block. When Oracle decides to move the updated row to a new data block, a row migration happens. Migrated or chained rows forces Oracle to access several blocks for a row. So, the need for more I/O increases. And when there are many of these chained or migrated rows in a table or tables, it can put your database to a crawl. One possible fix is to break the table into 2 or more tables with a one-to-one relationship. But you have to test the queries involved to get an acceptable query speed. If you already know which query is slow, you can jump to Tip #3 below. But if your entire website or system is slow and you don't know which of the queries are slow, you have to hunt for it. Given the appropriate licenses, you have access to Oracle's automatic tuning tools. These tools can help you spot slow queries and even give you recommendations. But if manual tuning is your only choice, you can use the V$SQL and related views. Then, spot the slow ones that are still in the shared pool. Here's one sample code to do that: SELECT * FROM V$SQL WHERE EXECUTIONS > 0 ORDER BY ELAPSED_TIME DESC; The output of the above query will have the slowest query on top. The list will include other queries that ran in the background. So, the trick is to inspect the value of the SQL_TEXT column and look for a familiar query that you made. Once you find a suspected query, proceed to the next tip. When you tell Oracle to run a SQL statement, you don't tell Oracle how to do it. You only provide what you need, and Oracle does the "how" for you. In other words, when you run a SELECT statement, you want columns and rows given to you. How it will be given to you is Oracle's job. And that's why you need to see how Oracle does it for you. That's where the execution plan comes into the picture. It contains a series of steps on how Oracle will process your query. It happens when Oracle receives your command for the first time. Oracle parses your query and checks for syntax and semantics. Then, it will generate a hash value for your SQL statement and store the plan in the shared pool. So, the next time you run the exact query, it won't formulate a plan anymore. Generating the best plan is done by the query optimizer. The optimizer uses statistics to calculate costs. It gives a cost to every plan and chooses the one with the lowest cost. Since Oracle has the internal statistics and tools at its disposal, the optimizer is usually in a better position to decide how to run our query. To see the execution plan for a SQL statements, you can use the EXPLAIN PLAN. Here's the syntax: EXPLAIN PLAN FOR ; The is the query you want to check. Let's have an example. The sample query we want to be explained is to get some sales information about OR number 63091. It includes the name of the customer, the products bought, quantity, and price. EXPLAIN PLAN FOR SELECT sh.OR_NUMBER ,sh.SALES_DATE ,c.CUSTOMER_NAME ,p.PRODUCT_NAME ,sd.QUANTITY ,sd.TOTAL_PRICE FROM SALES_HEADER sh INNER JOIN CUSTOMER c ON SH.CUSTOMER_ID = c.CUSTOMER_ID INNER JOIN SALES_DETAIL sd ON SH.SALES_HEADER_ID = sd.SALES_HEADER_ID INNER JOIN PRODUCT p ON sd.PRODUCT_ID = p.PRODUCT_ID WHERE OR_NUMBER='63091'; SELECT * FROM TABLE(dbms_xplan.display()); Running EXPLAIN PLAN will let Oracle store the steps in PLAN_TABLE. One way to view it is to SELECT the PLAN_TABLE. But to have a better view, you need to query the DBMS_XPLAN.DISPLAY(). The output is the same for any tool. Here's the output in dbForge Studio for Oracle. The optimizer chose several steps for our query. Note that the SALES_HEADER has 5,000 rows and SALES_DETAIL has 15,000 rows. While the PRODUCT table has 20 rows. The result of the query is 4 rows. A plan will show the join order, access path, and join method used. The tree format implies that an indented step is part of a higher step. Notice that the optimizer used nested loops, hash join, buffer sort, and more. Each has a corresponding cost and time. The TQ and IN-OUT columns with values show Oracle used parallel execution to improve query speed further. If this makes you scratch your head and you don't spot any problems, please read the next subsection. The plan needs interpretation to spot problems in it. So, for starters, what are some of the problem areas you need to watch out for? Full table scans – This means that Oracle needs to scan each row of the whole table sequentially. This is not always bad. But when your table is large to get a few results, this is not good. It's a sign that the query didn't use an index. Yet the optimizer may still choose to scan the whole table if it sees it to be the faster option. But your users' patience may not align with it. To spot a table scan in an execution plan, look for TABLE ACCESS FULL. The ideal is an INDEX UNIQUE SCAN or an INDEX RANGE SCAN of a few rows. Too many rows scanned – This is related to the table scan. If the number of rows scanned is too many and the result is few, then an index is missing. The Rows column in the execution plan will give you an idea. Sorting – arranging the rows frequently is an expensive process. If your queries are slow with ORDER BY but fast without it, consider an index on the ORDER BY columns. Cost – The optimizer assigns costs to each query and its parts. The higher the number the slower the query. Your goal is to lower it by simplifying your query and creating appropriate indexes. So, how does the plan above fare with the above points? Though the query we used ran at 00:00:01, it still has a problem. And the growing data will make it worse by the day. So, what happened in the query based on the EXPLAIN PLAN? Notice the TABLE ACCESS STORAGE FULL on the SALES_DETAIL table? It means Oracle made a sequential scan on the whole table. The rows accessed by the table scan are 15,000 rows. That's the entire table. The join between it and SALES_HEADER used a table scan to return rows with sales date, product, quantity, and price. The result set is only 4 rows. Does Oracle need to scan 15,000 rows for a result set of 4? There's no index used to join using the SALES_HEADER_ID column. Let's stop at this point for the next tip. But you will see the solution in one of the examples later. The optimizer uses statistics on database objects involved in a query. This includes table, column, index, and system statistics. Without it, the optimizer may choose a less optimal plan. Oracle performs automatic updates on optimizer statistics using the DBMS_STATS. The default interval is 15 minutes. But on high-demand scenarios where updates, inserts, and deletes happen so fast, it may not be able to keep up. You can reconfigure automatic statistics collection. Or you may choose to run DBMS_STATS yourself to update a specific database object. Here's the syntax to update table, column, and index statistics: EXEC DBMS_STATS.GATHER_TABLE_STATS('your_schema', 'your_table_name', CASCADE => TRUE); A good tool to analyze query performance is not just a pretty face. It should be simple and let you spot problems faster. So, you can fix query performance in a nick of time. Devart's family of products includes dbForge Studio for Oracle. And you need to check out its Query Profiler. The Query Profiler also has an EXPLAIN PLAN. In addition, it has Session Statistics that affect query performance. The productivity booster here is the side-by-side performance comparison. With it, you can compare 2 or more executions based on the changes you make to the query and other fixes you add, like indexes. Let me show it to you using the query we had earlier. First, you need to turn the Query Profiling Mode ON from the toolbar. Then, run the query. After that, you will see a profiling summary with total execution time, connection info, and more. Check it out below: Then, you can click the Plan for this query profile and see the EXPLAIN PLAN in tree format. The difference between the default EXPLAIN PLAN with this is the step# so you will see which of the steps ran first and what's next. Finally, clicking Session statistics will show the following: In Session Statistics, you will see more information about table scans, sorts, physical and logical reads, and more. Later, all of these will make sense when we compare 2 query profiles. So, please continue reading. For now, let's proceed with some real-world examples. It's time to go back to the example earlier. The obvious problem that caused a table scan is a missing index. So, let's add an index on SALES_HEADER_ID in the SALES_DETAIL table. Let's also cover the columns QUANTITY and TOTAL_PRICE because the query includes these. Here's the DDL command: CREATE INDEX IX_SALES_DETAIL_SALES_HEADER_ID ON SALES_DETAIL (SALES_HEADER_ID, QUANTITY, TOTAL_PRICE); Now, using dbForge Studio for Oracle, let's turn on the Query Profiling Mode and re-run the query. After getting a second query profile, I renamed the 2 query profiles with "Before Adding Index" and "After Adding Index" respectively. So, we can identify which profile has the fix or not. Then, I press CTRL and click the 2 profiles. Check out the results below: The index did a good job of reducing execution time. Let's check the Plan next. We can't have a side-by-side comparison of the EXPLAIN PLAN. But as you can see below, the 24 steps earlier are now reduced to 19 steps after adding the index. Check out the Before Adding Index Plan first. Then, check out the total steps and the use of the new index below. The table scan of 15,000 rows is now gone. Finally, let's check the side-by-side comparison of Session Statistics. You've heard that SELECT * is slow. Today, you will prove it. Let's compare the query profiles of the two queries below: -- Using SELECT * SELECT * FROM SALES_HEADER sh INNER JOIN SALES_DETAIL sd ON SH.SALES_HEADER_ID = SD.SALES_HEADER_ID WHERE sh.SALES_HEADER_ID = 100; -- Choosing only the needed columns SELECT sh.OR_NUMBER ,sh.SALES_DATE ,sh.CUSTOMER_ID ,sd.PRODUCT_ID ,sd.QUANTITY ,sd.TOTAL_PRICE FROM SALES_HEADER sh INNER JOIN SALES_DETAIL sd ON SH.SALES_HEADER_ID = SD.SALES_HEADER_ID WHERE sh.SALES_HEADER_ID = 100; The difference between the 2 queries is the columns to include. Nothing more. Here's the comparison of their execution time: See? SELECT * is slower. It ran more than 2 seconds while the other ran only a fraction of a second. The EXPLAIN PLAN reveals why. First, let's have the plan for SELECT *: The above is the EXPLAIN PLAN of our second query. For now, take note of the final bytes in step #12. Here's the EXPLAIN PLAN of using a select list of columns. Compare the above image from the previous plan. The bytes consumed are higher with SELECT *. Using a list of columns in SELECT is worth it. So, avoid SELECT * if you don't need all the columns. Let's examine 2 almost identical SELECT statements. The only difference is in the WHERE clause. The OR_NUMBER column of the SALES_HEADER table is a VARCHAR column. So, you need to enclose a literal value with quotes. But if you don't use quotes, Oracle will do an implicit conversion without you knowing it. The effect is a table scan instead of using an index. Here are the 2 queries: SELECT sh.OR_NUMBER ,sh.SALES_DATE ,c.CUSTOMER_NAME ,p.PRODUCT_NAME ,sd.QUANTITY ,sd.TOTAL_PRICE FROM SALES_HEADER sh INNER JOIN CUSTOMER c ON SH.CUSTOMER_ID = c.CUSTOMER_ID INNER JOIN SALES_DETAIL sd ON SH.SALES_HEADER_ID = sd.SALES_HEADER_ID INNER JOIN PRODUCT p ON sd.PRODUCT_ID = p.PRODUCT_ID WHERE OR_NUMBER='63091'; -- With quotes SELECT sh.OR_NUMBER ,sh.SALES_DATE ,c.CUSTOMER_NAME ,p.PRODUCT_NAME ,sd.QUANTITY ,sd.TOTAL_PRICE FROM SALES_HEADER sh INNER JOIN CUSTOMER c ON SH.CUSTOMER_ID = c.CUSTOMER_ID INNER JOIN SALES_DETAIL sd ON SH.SALES_HEADER_ID = sd.SALES_HEADER_ID INNER JOIN PRODUCT p ON sd.PRODUCT_ID = p.PRODUCT_ID WHERE OR_NUMBER=63091; -- NO QUOTES for a VARCHAR column. Treated as a number. The EXPLAIN PLAN is enough to see that when Oracle does implicit conversion, a table scan results and it's bad. Here's the execution plan for a quoted OR_NUMBER. It used the index, and the query cost is 1. Now, check the EXPLAIN PLAN of the VARCHAR column with no quotes on the literal value. A table scan results when Oracle converts a number to VARCHAR. The query cost is also higher (The cost is 14 compared to 1 earlier). This happens in the background, and you won't know until you do some performance tuning. The trick is to evaluate values and columns with the same data type. You may need to cast it to the same data type in certain cases. Well, the following are not so secret if you practice them already. But they are not the usual cup of tea. If you are already a seasoned Oracle developer or app developer, you can skip this part. A database maintains an invisible index, but the optimizer will ignore it unless explicitly specified with a hint. You can make an existing index invisible and see if there's a negative impact on performance before dropping it. Or you can create a new invisible index and with a query hint, you can test if performance will improve. Just add the INVISIBLE keyword to ALTER INDEX. ALTER INDEX index_name INVISIBLE; Or add the INVISIBLE keyword to CREATE INDEX. CREATE INDEX index_name ON TABLE(COLUMN[S]) INVISIBLE; Example of using a hint to use an invisible index: ALTER INDEX IX_SALES_DETAIL_SALES_HEADER_ID INVISIBLE; SELECT /*+ use_invisible_indexes INDEX (SALES_DETAIL IX_SALES_DETAIL_SALES_HEADER_ID) */ * FROM SALES_DETAIL sd WHERE SD.SALES_HEADER_ID = 100; Check the EXPLAIN PLAN to see if there's an improvement. The Results Cache stores the results of a query. It reduces redundant computations for identical queries and enhances overall performance. Example: SELECT /*+ RESULT_CACHE */ * FROM PRODUCT p; If your app uses data that updates less frequently, you can cache them for faster response. Outside factors like your application design influence your Oracle database performance. Here are some useful tips: Use connection pooling. Connecting to a database is an expensive operation that is not scalable. It's not good if your app frequently connects and disconnects from the database. Or allows many concurrent connections. Pool connections to minimize connections. Parse SQL statements once and execute them many times. Parsing SQL statements and creating execution plans is also an expensive operation. Design your app so that execution plans are reused many times from the shared pool. Effective use of bind variables. To ensure the reuse of SQL statements and their plans, use bind variables and not literal values. For example, instead of using WHERE LASTNAME='Smith' (use of literal value), use WHERE LASTNAME = :1 (bind variable). Let's recap. To get started with performance tuning, you need to know the basics of Oracle database architecture and know SQL well. Then, we learned about a lean Oracle database and avoiding chained and migrated rows. We also learned the basics of interpreting the EXPLAIN PLAN results. And you saw some examples of slow queries and their fix. Oracle performance tuning involves other aspects. The official documentation has a lot to mention about it. But we covered the basics here. You can master this in time with practice and a dash of determination. Reading the official documentation can become a lifesaver one day. So, put that in your list of To-dos. Execution plans may change too as data grows. So, monitor if the previous plan still performs well. Then, adjust accordingly. We also presented a brilliant Oracle GUI tool called dbForge Studio for Oracle. Why not give dbForge Studio for Oracle a try to boost your productivity in crushing Oracle performance bottlenecks today? Last modified: January 16, 2024 A well-planned methodology is the key to success in performance tuning. Without a proper diagnosis of the issue any troubleshooting step taken will further delay the resolution. Example, a DBA can not just blindly start tracing/tuning a SQL/Sessions when all symptoms on careful analysis points towards issues with DIsk/Memory/Network etc. Also remember that most common performance gains are attained through SQL/Code/application tuning and increasing hardware resources should always be your last resort. Developers play an important role in writing applications with effective SQL statements so it is important that DBA and Developers should work together when facing a database/application issue. When DBAs are reported a performance issue, first step is to get as much information as possible that is related to the issue. You can ask below example preliminary questions to users/developer to collect the first level of information.. 1. What operations/program are executed? 2. Is it Oracle seeded or custom program? 3. How much time it used to take earlier? 4. Is the run time increased over time or you are seeing sudden increase in run time? 5. Was there any recent code change/migration? 6. Is it always slow or for certain time of the day only? 7. Is it slow for all parameters or for some specific parameters? 8. How much data is getting processed? 9. What is the frequency of job execution? Was there any change in frequency? 10. Does the problem happens on both their test and production systems? Asking above kind of questions will help you in deciding what part of system you should target. Target the whole system Target a single session Target a single SQL statement Your goal should be to answer below three questions: Where is time spent? You need to identify where the time is getting consumed. Is it the code or network latency or disk I/O bottlenecks or the CPU/Memory crunch How is time spent? You need to find out how the time is spent and how much time is spend in each layer – database, application, network, disk etc . How to reduce the time spent? Finally, based on above information see where the major chunk of time is spent and how to reduce it. High level steps for System level performance issues Use vmstat top/prstat to identify system wide CPU/Memory consumption. Use iostat to verify if disks are the bottleneck Use netstat/tnsping etc to verify if network is issue. Verify if any other resource intensive processes are running on server. Verify filesystem space. Check alert logs, application logs, traces etc. Check database locks Generate AWR reports to see what is eating up resources. Check if increasing application/database memory/redo/undo/temp/SGA will help. High level steps for Session level performance issues Find the Top SQLs executing under session. Apply SQL optimization techniques on top SQLs. Verify locking at session level Generating AWR/ASH for that duration may help in providing useful information. High level steps for SQL level performance issues Apply trace, Generate TKPROF file and analyze for top resource consuming SQL. Avoid full table scans on large tables Possible indexes on columns contained in the WHERE clause Use AWR/ASH reports to get collective information Also use SQLTRPT/SQLT Verify is statistics are current Verify if indexes are valid and are adequate. Verify is parallelism/materialized views/Baselines/SQL Profiles will help Monitor V$SESSION_LONGOPS to detect long running operations Decide on using Hints if those are helping. Table partitioning can be thought of as an option based on kind and size of tables. This is one of the post in Performance Tuning Fundamentals Series. Click on below links to read more posts from the series: Sql statements are written to retrieve /fetch data from the database. We want our sql statement to run fast(sql tuning) and provide the results in secs.A poorly designed sql could slow down the entire database operation and bring the entire operation to halt. It is much harder to write efficient SQL than it is to write functionally correct SQL. sql performance tuning can significantly improve a system's health and performance.The key to tuning SQL is to minimize the data it accesses to provide the result. We can minimize the data it accesses to deliver the result through an optimal search path. A simple example would be select * from dept where emp=10 Now this query will have to search the entire table dept to find out the data where emp=10. So it has to access the full tableNow if we create the index to the emp column, then it can just access the index and get the result. So here it accesses the fewer data Here are the general tips for sql performance tuning (1) First you must have all the required tools for sql tuning. You must have good information on tracing, formatting the trace, explain plan, and reading the explain plan in oracle.Good knowledge about the various join method available in Oracle and how to use them efficiently (2) Read fewer data and be I/O efficient.The more data you read for the sql statement, the more latches it needs to acquire and it slows down the performance. so it must be doing fewer logical reads alwaysWrite a sensible sql statement with the proper filters. Do check about the number of rows in various tables involved and find out the best method to create the sql statement See also How to import statistics in Oracle(2) Use good Oracle indexesB-Tree indexes and Bitmap indexes can be used to increase the performance of the queries if the data returned is less than 10%. But we must be careful while creating the index as it needs to be maintained for insert, update, and delete operations also. So creating an index creates overhead over many things. So we must carefully examine the effect of creating the index. (3) Avoid sql which disables the usage of index SQL that Disables Indexes(a)Functions (to_char(), to_date(), etc.)Fix: move the function to the "constant/bind variable" side(b) Type CastingIn SQLwhere emp_no = 10 (emp_no is a varchar2)In PL/SQLwhere emp_no = v_emp_num (v_emp_num is a number)Modifiersand id + 0 = 111and date + 1 = sysdate (try date = sysdate – 1)Fix: Change it to avoid it (4) Always use bind variables in the application. If you dont use bind variable in oracle , the sql will be parsed every time and will impact the database performance. If it contains the bind variable, sql would be cached and further execution will not require parsing thus overall performance of the system is improved (5) UNION vs OR. Use UNION for queries with two clear execution paths; each returning a relatively small number of rows. Don't use a union for queries that are likely to return many rows since all rows need to be sorted and most of them will be discarded. OR tends to disable the index (6) Use the accurate optimizer statistics on the table to get the optimal plan. See also How to Update the Variables in Context file in EBS(7) If you are using a function on the expression on the condition, check if there is a function-based index on that column. If it is not present the index will not be used (8) Use exists vs in and Non exists vs not in for correlated subqueries (9) Avoid Poor coding practicesSome Tips(a) Avoid Cartesian join. Make sure all the tables required in the queries are needed and are linked to each other(b) Use Decode to avoid multiple trips to the database(c) Try to avoid outer join(d) Sometimes decomposing the logic into small parts makes the work faster (10) If trying to use the complex view, check if the base tables can be used instead as the view tends to make the performance bad (11) Use UNION ALL Vs UNION if you know that the data retrieved will not have duplicate rows (12) Use hints to optimize the execution plan. Sometimes hints can be used to change the execution plan for the query so as to take the most optimal path.Sometimes bind peeking create a bad plan, so in that case, putting necessary hint to fix the plan help in getting a good performance every timeThe most common hints are/*+ LEADING (table alias) */ specifies table by which to start join/*+ FIRST_ROWS */ very good for online screens – favors NESTED LOOPS/*+ INDEX (table alias.index name) */ specifies index you want to use. Note: if index gets dropped and recreated and name changes then hint is no longer valid./*+ USE_NL (table alias1 table alias 2)*/ asks optimizer to use the Nested Loop Join for the two tables specified See also What is PLSQL records in OracleAvoid unnecessary optimizer hint and use them with care These are some of the tips to avoid issues and do the sql tuning. Sql tuning is a big Ocean and you can learn things by practicing only. Best Of Luck!! Also Reads Workload Repository(AWR)Hanganalyze and system state dumpOracle waits eventsHow to find which sid is doing full table scan Here are the Important Oracle Database Health check Database, Datafile and Backup select name,open_mode,log_mode,database_role,force_logging,flashback_on from v$database;select * from v$recover_file;select distinct status,count() from v$datafile group by... opmnctl is widely used in iAS administration,We should know all the useful command for its administration. Here I am talking about some Opmnctl command every administrator... This is part of Oracle SQL tutorial and has good examples, explanation on single row functions in Oracle Sql Functions are programs that take zero... The ODF Comparison utility(ADODFCMP) compares the data model of a customer's data to a standard set of data model files from the current Oracle Application... 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. Perform the following initial standard checks: 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. 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. 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. 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. 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". 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. 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. Page 2 If you use the Database Configuration Assistant (DBCA) to create a database, then the seed database automatically includes the necessary tablespaces. If you choose not to use DBCA, then you must create extra tablespaces after creating the database. All databases should have several tablespaces in addition to the SYSTEM and SYSAUX tablespaces. These additional tablespaces include: A temporary tablespace, which is used for operations such as sorting An undo tablespace to contain information for read consistency, recovery, and undo statements At least one tablespace for application use (in most cases, applications require several tablespaces) For extremely large tablespaces with many data files, you can run multiple ALTER TABLESPACE ... ADD DATAFILE statements in parallel. During tablespace creation, the data files that make up the tablespace are initialized with special empty block images. Temporary files are not initialized. Oracle Database does this to ensure that it can write all data files in their entirety, but this can obviously be a lengthy process if done serially. Therefore, run multiple CREATE TABLESPACE statements concurrently to speed up tablespace creation. For permanent tables, the choice between local and global extent management on tablespace creation can greatly affect performance. For any permanent tablespace that has moderate to large insert, modify, or delete operations compared to reads, choose local extent management. Permanent Tablespaces - Automatic Segment-Space Management For permanent tablespaces, Oracle recommends using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management. Temporary Tablespaces Properly configuring the temporary tablespace helps optimize disk sort performance. Temporary tablespaces can be dictionary-managed or locally managed. Oracle recommends the use of locally managed temporary tablespaces with a UNIFORM extent size of 1 MB. You should monitor temporary tablespace activity to check how many extents the database allocates for the temporary segment. If an application extensively uses temporary tables, as in a situation when many users are concurrently using temporary tables, then the extent size could be set smaller, such as 256K, because every usage requires at least one extent. The EXTENT MANAGEMENT LOCAL clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The default for SIZE is 1M. A temporary tablespace can grow to a very large size due to spikes in temp usage by queries. Sorts, hash joins, and query transformations are examples that might cause high temp usage. Automatic Temp Tablespace Sizing is a feature which shrinks the temporary tablespace in the background when temp usage has subsided. In addition, if temp usage is increasing, the feature would pre-emptively grow the temporary tablespace to ensure query performance is not impacted. This feature alleviates the need for a DBA to manually size the temporary tablespace. These are the two scenarios which would benefit: Shrinking the temporary tablespace to reclaim unused space. Growing the temporary tablespace in anticipation of high temp usage. Page 3 Measuring Database Performance Gathering Database Statistics Automatic Performance Diagnostics Comparing Database Performance Over Time Analyzing Sampled Data Instance Tuning Using Performance Views Page 4 Any session that is connected to the database and is waiting for an event that does not belong to the Idle wait class is considered an active session. Oracle Database samples active sessions every second and stores the sampled data in a circular buffer in the shared global area (SGA). The sampled session activity is accessible using the V$ACTIVE_SESSION_HISTORY view. Each session sample contains a set of rows and the V$ACTIVE_SESSION_HISTORY view returns one row for each active session per sample, starting with the latest session sample rows. Because the active session samples are stored in a circular buffer in the SGA, the greater the system activity, the smaller the number of seconds of session activity that can be stored. This means that the duration for which a session sample is displayed in the V$ view is completely dependent on the level of database activity. Because the content of the V$ view can become quite large during heavy system activity, only a portion of the session samples is written to disk. By capturing only active sessions, a manageable set of data can be captured with its size being directly related to the work being performed, rather than the number of sessions allowed on the system. Active Session History (ASH) enables you to examine and perform detailed analysis on both current data in the V$ACTIVE_SESSION_HISTORY view and historical data in the DBA_HIST_ACTIVE_SESS_HISTORY view, often avoiding the need to replay the workload to trace additional performance information. ASH also contains execution plan information for each captured SQL statement. You can use this information to identify which part of SQL execution contributed most to the SQL elapsed time. The data present in ASH can be rolled up in various dimensions that it captures, including: SQL identifier of SQL statement SQL plan identifier and hash value of the SQL plan used to execute the SQL statement SQL execution plan information Object number, file number, and block number Wait event identifier and parameters Session identifier and session serial number Module and action name Client identifier of the session Service hash identifier Consumer group identifier You can gather this information over a specified duration into an ASH report. Active session history sampling is also available for Active Data Guard physical standby instances and Oracle Automatic Storage Management (Oracle ASM) instances. On these instances, the current session activity is collected and displayed in the V$ACTIVE_SESSION_HISTORY view, but not written to disk. Page 5 Starting with Oracle Database 12c Release 2 (12.2), Automatic Workload Repository (AWR) data can be captured for Active Data Guard (ADG) standby databases. This feature enables analyzing any performance-related issues for ADG standby databases. AWR snapshots for ADG standby databases are called remote snapshots. A database node, called destination, is responsible for storing snapshots that are collected from remote ADG standby database nodes, called sources. A destination can be either an ADG primary database or a non-ADG database. If a destination is an ADG primary database, then it is also a source database, and its snapshots are local snapshots. A source is identified by a unique name or source name by which it is known to a destination. You can assign a name to a destination node or a source node during its configuration. Otherwise, the value of the initialization parameter DB_UNIQUE_NAME is assigned as a name for a node. Each source must have two database links, a destination-to-source database link and a source-to-destination database link. These database links are configured for each source during the ADG deployment. You must manually reconfigure these database links after certain ADG events, such as failovers, switchovers, and addition and removal of hosts, so that the database applications continue functioning properly after these events. You can take the remote snapshots either automatically at scheduled time intervals or manually. The remote snapshots are always started by the destination node. After the destination initiates the snapshot creation process, sources push their statistics data to the destination using database links. The snapshot data on the destination can be accessed using AWR reports, Oracle Database import and export functions, and user-defined queries. The Automatic Database Diagnostic Monitor (ADDM) application can use the AWR data for analyzing any database performance-related issues. Destination Database Responsibilities A destination database manages the following tasks: Registering sources Assigning unique identifier for each source Creating database links between destination and sources Scheduling and initiating automatic snapshots for sources Managing destination workload by coordinating snapshots among sources Managing snapshot settings for each source Assigning identifiers to newly generated snapshots Partitioning the AWR tables Storing the performance data in the local AWR Purging the AWR data of destination and sources Source Database Responsibilities A source database manages the following tasks: Storing its performance data in the local AWR Sending its AWR data to the destination Responding to service requests from the destination Extracting the AWR data from the destination The Remote Management Framework (RMF) is an architecture for capturing performance statistics (AWR data) in an Oracle database. The RMF topology is a centralized architecture that consists of all the participating database nodes along with their metadata and connection information. The RMF topology has one database node, called destination, which is responsible for storing and managing performance data (AWR data) that is collected from the database nodes, called sources. A candidate destination is a source that can be configured in such way that it can replace the original destination, when the original destination is unavailable or is downgraded. A topology can have only one destination, and one or more candidate destinations. Each database node in a topology must be assigned a unique name. This can be done using the procedure DBMS_UMF.configure_node() during configuring a node. If the name for a node is not provided in this procedure, then the value of the initialization parameter DB_UNIQUE_NAME is used as the name for a node. The database nodes in a topology communicate with each other using database links. The database links between destination and sources are created either manually or by the ADG deployment. A service is an application running on a topology. For example, an AWR service running on a topology enables remote AWR snapshots for all the database nodes in that topology. The RMF APIs are the PL/SQL procedures and functions that can be used to configure the RMF topology. The RMF APIs are declared in the PL/SQL package DBMS_UMF. Setting Up the RMF Topology You need to set up the RMF topology for collecting performance statistics for an Oracle database. The following are the prerequisites for setting up the RMF topology: You must create destination to source and source to destination database links for all the database nodes to be registered in the RMF topology. This setup should be done during the ADG deployment. The following are the steps for setting up the RMF topology: Configure database nodes to add to the topology. Create the topology. Register database nodes with the topology. (Optional) Register database links between the nodes in the topology. This configuration is required when a destination becomes unavailable and a candidate destination needs to connect to the remaining nodes in the topology using database links. Example for Setting Up the RMF Topology In this example, the three database nodes T, S0, and S1 are added to the topology Topology_1. Node T is the destination node and nodes S0 and S1 are the source nodes. Node S1 is a candidate destination, that is, when the original destination T is not available, node S1 becomes the new destination. The AWR service is enabled for all the sources in the topology. Assume that the following database links are already created during the ADG deployment: DBLINK_T_to_S0: Database link from T to S0. DBLINK_T_to_S1: Database link from T to S1. DBLINK_S0_to_T: Database link from S0 to T. DBLINK_S0_to_S1: Database link from S0 to S1. DBLINK_S1_to_T: Database link from S1 to T. DBLINK_S1_to_S0: Database link from S1 to S0. The following is a sample code for setting up the RMF topology:/* Configure the nodes T, S0, and S1 by executing these procedures */ /* Execute this procedure on node T */ SQL> exec DBMS_UMF.configure_node ('T'); /* Execute this procedure on node S0 */ SQL> exec DBMS_UMF.configure_node ('S0', 'DBLINK_S0_to_T'); /* Execute this procedure on node S1 */ SQL> exec DBMS_UMF.configure_node ('S1', 'DBLINK_S1_to_T'); /* Execute the following procedures on the destination node T */ /* Create the topology 'Topology_1' */ SQL> exec DBMS_UMF.create_topology ('Topology_1'); /* Register the node S0 with the topology 'Topology_1' */ SQL> exec DBMS_UMF.register_node ('Topology_1', 'S0', 'DBLINK_T_to_S0', 'DBLINK_S0_to_T', 'TRUE' /* Set it as a source */, 'FALSE' /* Set it as not a candidate destination */); /* Register the node S1 with the topology 'Topology_1' */ SQL> exec DBMS_UMF.register_node ('Topology_1', 'S1', 'DBLINK_T_to_S1', 'DBLINK_S1_to_T', 'TRUE' /* Set it as a source */, 'TRUE' /* Set it as a candidate destination */); /* Register the database links between the nodes S0 and S1 in the topology 'Topology_1'. * When destination T is unavailable at the time of failover, the source S0 can connect to * the candidate destination S1 using this database link. */ SQL> exec DBMS_UMF.create_link ('Topology_1', 'S0', 'S1', 'DBLINK_S0_to_S1', 'DBLINK_S1_to_S0'); /* Enable the AWR service on the node S0 in the topology 'Topology_1' */ SQL> exec DBMS_WORKLOAD_REPOSITORY.register_remote_database(node_name=>'S0'); /* Enable the AWR service on the node S1 in the topology 'Topology_1' */ SQL> exec DBMS_WORKLOAD_REPOSITORY.register_remote_database(node_name=>'S1'); Managing ADG Role Transition An ADG role transition occurs when the ADG Primary or original destination fails (failover event) or when an ADG standby database or candidate destination takes over the role of the ADG Primary during the maintenance phase (switchover event). Oracle recommends that you perform the following configuration steps before making the role change, that is, before making the candidate destination as the new destination due to the failover or switchover event: Create database links between the sources by executing the following procedure on each source:SQL> EXEC DBMS_UMF.CREATE_LINK (topology name, source name, candidate destination name, source to candidate destination database link, candidate destination to source database link); Take an AWR snapshot on the candidate destination. Restart the candidate destination as well as all the sources. After completing the preceding configuration steps, you can make the role change by executing the following procedure on the candidate destination:SQL> EXEC DBMS_UMF.SWITCH_DESTINATION(topology name, force_switch=>FALSE); Getting the Details of Registered RMF Topologies The RMF views described below show the configuration information about all the registered RMF topologies in a multi-database environment. Table 6-3 RMF Views RMF View Description DBA_UMF_TOPOLOGY Shows all the registered topologies in a multi-database environment. Each topology has a topology name, a destination ID, and topology state. To enable RMF, the topology state of at least one topology should be ACTIVE. DBA_UMF_REGISTRATION Shows all the registered nodes in all the topologies in a multi-database environment. DBA_UMF_LINK Shows all the registered database links in all the topologies in a multi-database environment. DBA_UMF_SERVICE Shows all the registered services in all the topologies in a multi-database environment. You can view the AWR data stored in the ADG standby databases using Oracle supplied AWR views and AWR reports. Viewing AWR Data Using AWR Views You can view the historical data stored in AWR using the DBA_HIST views described in the section "Using Automatic Workload Repository Views". Viewing AWR Data Using AWR Reports You can view the performance statistics related to ADG standby databases using AWR reports. The primary interface for generating AWR reports is Oracle Enterprise Manager Cloud Control (Cloud Control). Whenever possible, generate AWR reports using Cloud Control. If Cloud Control is unavailable, then generate AWR reports using the Oracle supplied SQL scripts. The DBA role is required to run these scripts. The AWR data can be queried for a particular source using the source name-mapped database ID pair. The mapped database ID is similar to the database identifier (DBID) that is used by AWR to identify a database instance and is stored in the DBID column in the AWR tables. The AWR DBID value is derived as follows for the ADG standby databases: For a destination, the AWR DBID value is the value of V$DATABASE.CON_DBID. For a source, the AWR DBID value is the value of DBMS_UMF.GET_NODE_ID_LOCAL() or the value of the column NODE_ID in the DBA_UMF_REGISTRATION view. As snapshot IDs are not unique across sources, the pair of snapshot ID-mapped database ID identifies a snapshot for a particular source. Page 6 Example 7-1 Example ADDM ReportFINDING 1: 31% impact (7798 seconds) -------------------------------------- SQL statements were not shared due to the usage of literals. This resulted in additional hard parses which were consuming significant database time. RECOMMENDATION 1: Application Analysis, 31% benefit (7798 seconds) ACTION: Investigate application logic for possible use of bind variables instead of literals. Alternatively, you may set the parameter "cursor_sharing" to "force". RATIONALE: SQL statements with PLAN_HASH_VALUE 3106087033 were found to be using literals. Look in V$SQL for examples of such SQL statements. In Example 7-1, the finding points to a particular root cause, the usage of literals in SQL statements, which is estimated to have an impact of about 31% of total DB time in the analysis period. The finding has a recommendation associated with it, composed of one action and one rationale. The action specifies a solution to the problem found and is estimated to have a maximum benefit of up to 31% DB time in the analysis period. Note that the benefit is given as a portion of the total DB time and not as a portion of the finding's impact. The rationale provides additional information on tracking potential SQL statements that were using literals and causing this performance issue. Using the specified plan hash value of SQL statements that could be a problem, a DBA could quickly examine a few sample statements. When a specific problem has multiple causes, ADDM may report multiple problem and symptom findings. In this case, the impacts of these multiple findings can contain the same portion of DB time. Because the performance issues of findings can overlap, the sum of the impacts of the findings can exceed 100% of DB time. For example, if a system performs many reads, then ADDM might report a SQL statement responsible for 50% of DB time due to I/O activity as one finding, and an undersized buffer cache responsible for 75% of DB time as another finding. When multiple recommendations are associated with a problem finding, the recommendations may contain alternatives for solving the problem. In this case, the sum of the recommendations' benefits may be higher than the finding's impact. When appropriate, an ADDM action may have multiple solutions for you to choose from. In the example, the most effective solution is to use bind variables. However, it is often difficult to modify the application. Changing the value of the CURSOR_SHARING initialization parameter is much easier to implement and can provide significant improvement. Page 7 Generate an AWR Compare Periods report for Oracle RAC on a specific database using the command-line interface: At the SQL prompt, enter:@$ORACLE_HOME/rdbms/admin/awrgdrpi.sql Specify whether you want an HTML or a text report:Enter value for report_type: html In this example, an HTML report is chosen. A list of available database identifiers and instance numbers are displayed:Instances in this Workload Repository schema ~~~~~~ DB Id Inst Num DB Name Instance Host ----------- -------- ------------ ------------ ------------ 3309173529 1 MAIN main examp1690 3309173529 1 TINT251 tint251 samp251 3309173529 2 TINT251 tint252 samp252 3309173529 3 TINT251 tint253 samp253 3309173529 4 TINT251 tint254 samp254 Enter the values for the database identifier (dbid) and instance numbers (instance_numbers_or_all) for the first time period: Enter value for dbid: 3309173529 Using 3309173529 for Database Id for the first pair of snapshots Enter value for inst_num: 1,2 Using instances 1 for the first pair of snapshots Specify the number of days for which you want to list snapshot IDs in the first time period. Enter value for num_days: 2 A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed. Specify a beginning and ending snapshot ID for the first time period:Enter value for begin_snap: 102 Enter value for end_snap: 103 In this example, the snapshot with a snapshot ID of 102 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 103 is selected as the ending snapshot for the first time period. A list of available database identifiers and instance numbers are displayed:Instances in this Workload Repository schema ~~~~~~ DB Id Inst Num DB Name Instance Host ----------- -------- ------------ ------------ ------------ 3309173529 1 MAIN main examp1690 3309173529 1 TINT251 tint251 samp251 3309173529 2 TINT251 tint252 samp252 3309173529 3 TINT251 tint253 samp253 3309173529 4 TINT251 tint254 samp254 INSTNUM1 --- 1,2 Enter the values for the database identifier (dbid2) and instance numbers (instance_numbers_or_all2) for the second time period: Enter value for dbid2: 3309173529 Using 3309173529 for Database Id for the second pair of snapshots Enter value for instance_numbers_or_all2: 3,4 Specify the number of days for which you want to list snapshot IDs in the second time period. Enter value for num_days2: 1 A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed. Specify a beginning and ending snapshot ID for the second time period:Enter value for begin_snap2: 126 Enter value for end_snap2: 127 In this example, the snapshot with a snapshot ID of 126 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 127 is selected as the ending snapshot for the second time period. Enter a report name, or accept the default report name:Enter value for report_name: Using the report name awrracdiff_1st_1_2nd_1.html In this example, the default name is accepted and an AWR report named awrracdiff_1st_1_2nd_1.html is generated. Page 8 The Activity Over Time section is one of the most informative sections of the ASH report. This section is particularly useful for analyzing longer time periods because it provides in-depth details about activities and workload profiles during the analysis period. The Activity Over Time section is divided into 10 time slots. The size of each time slot varies based on the duration of the analysis period. The first and last slots are usually odd-sized. All inner slots are equally sized and can be compared to each other. For example, if the analysis period lasts for 10 minutes, then all time slots will 1 minute each. However, if the analysis period lasts for 9 minutes and 30 seconds, then the outer slots may be 15 seconds each and the inner slots will be 1 minute each. Each of the time slots contains information regarding that particular time slot, as described in Table 9-1. Table 9-1 Activity Over Time Column Description Slot Time (Duration) Duration of the slot Slot Count Number of sampled sessions in the slot Event Top three wait events in the slot Event Count Number of ASH samples waiting for the wait event % Event Percentage of ASH samples waiting for wait events in the entire analysis period When comparing the inner slots, perform a skew analysis by identifying spikes in the Event Count and Slot Count columns. A spike in the Event Count column indicates an increase in the number of sampled sessions waiting for a particular event. A spike in the Slot Count column indicates an increase in active sessions, because ASH data is sampled from active sessions only and a relative increase in database workload. Typically, when the number of active session samples and the number of sessions associated with a wait event increases, the slot may be the cause of a transient performance problem. To generate the ASH report with a user-defined slot size, run the ashrpti.sql script, as described in "Generating an ASH Report on a Specific Database Instance". Page 9 A latch is a low-level internal lock used by Oracle Database to protect memory structures. The latch free event is updated when a server process attempts to get a latch, and the latch is unavailable on the first attempt.There is a dedicated latch-related wait event for the more popular latches that often generate significant contention. For those events, the name of the latch appears in the name of the wait event, such as latch: library cache or latch: cache buffers chains. This enables you to quickly figure out if a particular type of latch is responsible for most of the latch-related contention. Waits for all other latches are grouped in the generic latch free wait event. Actions This event should only be a concern if latch waits are a significant portion of the wait time on the system as a whole, or for individual users experiencing problems. Examine the resource usage for related resources. For example, if the library cache latch is heavily contended for, then examine the hard and soft parse rates. Examine the SQL statements for the sessions experiencing latch contention to see if there is any commonality. Check the following V$SESSION_WAIT parameter columns: P1: Address of the latch P2: Latch number P3: Number of times process has slept, waiting for the latch Example: Find Latches Currently Waited ForSELECT EVENT, SUM(P3) SLEEPS, SUM(SECONDS_IN_WAIT) SECONDS_IN_WAIT FROM V$SESSION_WAIT WHERE EVENT LIKE 'latch%' GROUP BY EVENT; A problem with the previous query is that it tells more about session tuning or instant instance tuning than instance or long-duration instance tuning. The following query provides more information about long duration instance tuning, showing whether the latch waits are significant in the overall database time. SELECT EVENT, TIME_WAITED_MICRO, ROUND(TIME_WAITED_MICRO*100/S.DBTIME,1) PCT_DB_TIME FROM V$SYSTEM_EVENT, (SELECT VALUE DBTIME FROM V$SYS_TIME_MODEL WHERE STAT_NAME = 'DB time') S WHERE EVENT LIKE 'latch%' ORDER BY PCT_DB_TIME ASC; A more general query that is not specific to latch waits is the following:SELECT EVENT, WAIT_CLASS, TIME_WAITED_MICRO,ROUND(TIME_WAITED_MICRO*100/S.DBTIME,1) PCT_DB_TIME FROM V$SYSTEM_EVENT E, V$EVENT_NAME N, (SELECT VALUE DBTIME FROM V$SYS_TIME_MODEL WHERE STAT_NAME = 'DB time') S WHERE E.EVENT_ID = N.EVENT_ID AND N.WAIT_CLASS NOT IN ('Idle', 'System I/O') ORDER BY PCT_DB_TIME ASC; Table 10-3 Latch Wait Event Latch SGA Area Possible Causes Look For: Shared pool, library cache Shared pool Lack of statement reuse Statements not using bind variables Insufficient size of application cursor cache Cursors closed explicitly after each execution Frequent logins and logoffs Underlying object structure being modified (for example truncate) Shared pool too small Sessions (in V$SESSTAT) with high: parse time CPU parse time elapsed Ratio of parse count (hard) / execute count Ratio of parse count (total) / execute count Cursors (in V$SQLAREA/V$SQLSTATS) with: High ratio of PARSE_CALLS / EXECUTIONS EXECUTIONS = 1 differing only in literals in the WHERE clause (that is, no bind variables used) High RELOADS High INVALIDATIONS Large (> 1mb) SHARABLE_MEM cache buffers lru chain Buffer cache LRU lists Excessive buffer cache throughput. For example, inefficient SQL that accesses incorrect indexes iteratively (large index range scans) or many full table scans DBWR not keeping up with the dirty workload; hence, foreground process spends longer holding the latch looking for a free buffer Cache may be too small Statements with very high logical I/O or physical I/O, using unselective indexes cache buffers chains Buffer cache buffers Repeated access to a block (or small number of blocks), known as a hot block Sequence number generation code that updates a row in a table to generate the number, rather than using a sequence number generator Index leaf chasing from very many processes scanning the same unselective index with very similar predicate Identify the segment the hot block belongs to row cache objects Shared Pool and Library Cache Latch Contention A main cause of shared pool or library cache latch contention is parsing. There are several techniques that you can use to identify unnecessary parsing and several types of unnecessary parsing: This method identifies similar SQL statements that could be shared if literals were replaced with bind variables. The idea is to either: Manually inspect SQL statements that have only one execution to see whether they are similar:SELECT SQL_TEXT FROM V$SQLSTATS WHERE EXECUTIONS < 4 ORDER BY SQL_TEXT; Or, automate this process by grouping what may be similar statements. Estimate the number of bytes of a SQL statement that are likely the same, and group the SQL statements by this number of bytes. For example, the following example groups statements that differ only after the first 60 bytes.SELECT SUBSTR(SQL_TEXT, 1, 60), COUNT(*) FROM V$SQLSTATS WHERE EXECUTIONS < 4 GROUP BY SUBSTR(SQL_TEXT, 1, 60) HAVING COUNT(*) > 1; Or report distinct SQL statements that have the same execution plan. The following query selects distinct SQL statements that share the same execution plan at least four times. These SQL statements are likely to be using literals instead of bind variables.SELECT SQL_TEXT FROM V$SQLSTATS WHERE PLAN_HASH_VALUE IN (SELECT PLAN_HASH_VALUE FROM V$SQLSTATS GROUP BY PLAN_HASH_VALUE HAVING COUNT(*) > 4) ORDER BY PLAN_HASH_VALUE; Check the V$SQLSTATS view. Enter the following query: SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS FROM V$SQLSTATS ORDER BY PARSE_CALLS; When the PARSE_CALLS value is close to the EXECUTIONS value for a given statement, you might be continually reparsing that statement. Tune the statements with the higher numbers of parse calls. Identify unnecessary parse calls by identifying the session in which they occur. It might be that particular batch programs or certain types of applications do most of the reparsing. To achieve this goal, run the following query:SELECT pa.SID, pa.VALUE "Hard Parses", ex.VALUE "Execute Count" FROM V$SESSTAT pa, V$SESSTAT ex WHERE pa.SID = ex.SID AND pa.STATISTIC#=(SELECT STATISTIC# FROM V$STATNAME WHERE NAME = 'parse count (hard)') AND ex.STATISTIC#=(SELECT STATISTIC# FROM V$STATNAME WHERE NAME = 'execute count') AND pa.VALUE > 0; The result is a list of all sessions and the amount of reparsing they do. For each session identifier (SID), go to V$SESSION to find the name of the program that causes the reparsing. The output is similar to the following: SID Hard Parses Execute Count ------ ----------- ------------- 7 1 20 8 3 12690 6 26 325 11 84 1619The cache buffers lru chain latches protect the lists of buffers in the cache. When adding, moving, or removing a buffer from a list, a latch must be obtained. For symmetric multiprocessor (SMP) systems, Oracle Database automatically sets the number of LRU latches to a value equal to one half the number of CPUs on the system. For non-SMP systems, one LRU latch is sufficient. Contention for the LRU latch can impede performance on SMP computers with a large number of CPUs. LRU latch contention is detected by querying V$LATCH, V$SESSION_EVENT, and V$SYSTEM_EVENT. To avoid contention, consider tuning the application, bypassing the buffer cache for DSS jobs, or redesigning the application. The cache buffers chains latches are used to protect a buffer list in the buffer cache. These latches are used when searching for, adding, or removing a buffer from the buffer cache. Contention on this latch usually means that there is a block that is greatly contended for (known as a hot block). To identify the heavily accessed buffer chain, and hence the contended for block, look at latch statistics for the cache buffers chains latches using the view V$LATCH_CHILDREN. If there is a specific cache buffers chains child latch that has many more GETS, MISSES, and SLEEPS when compared with the other child latches, then this is the contended for child latch. This latch has a memory address, identified by the ADDR column. Use the value in the ADDR column joined with the X$BH table to identify the blocks protected by this latch. For example, given the address (V$LATCH_CHILDREN.ADDR) of a heavily contended latch, this queries the file and block numbers: SELECT OBJ data_object_id, FILE#, DBABLK,CLASS, STATE, TCH FROM X$BH WHERE HLADDR = 'address of latch' ORDER BY TCH; X$BH.TCH is a touch count for the buffer. A high value for X$BH.TCH indicates a hot block. Many blocks are protected by each latch. One of these buffers will probably be the hot block. Any block with a high TCH value is a potential hot block. Perform this query several times, and identify the block that consistently appears in the output. After you have identified the hot block, query DBA_EXTENTS using the file number and block number, to identify the segment. After you have identified the hot block, you can identify the segment it belongs to with the following query:SELECT OBJECT_NAME, SUBOBJECT_NAME FROM DBA_OBJECTS WHERE DATA_OBJECT_ID = &obj; In the query, &obj is the value of the OBJ column in the previous query on X$BH. The row cache objects latches protect the data dictionary. Page 10 Database Memory Allocation Tuning the System Global Area Tuning the Database Buffer Cache Tuning the Shared Pool and the Large Pool Tuning the Result Cache Tuning the Program Global Area Page 11 In automatic memory management, Oracle Database manages the system global area (SGA) and program global area (instance PGA) memory completely. This method is the simplest and is strongly recommended by Oracle. Before setting any memory pool sizes manually, strongly consider using automatic memory management. The only user-specified controls are the target memory size initialization parameter (MEMORY_TARGET) and optional maximum memory size initialization parameter (MEMORY_MAX_TARGET). Oracle Database tunes to the target memory size, redistributing memory as needed between the SGA and the instance PGA. The SGA can contain the In-Memory Column Store (IM column store) or memoptimize pool, both of which are optional. No matter which memory management method you use, size the IM column store with the INMEMORY_SIZE initialization parameter, and the memoptimize pool with the MEMOPTIMIZE_POOL_SIZE initialization parameter. The IM column store size are memoptimize pool are included in the memory target, but are not managed by the automatic resize algorithm. For example, if you set MEMORY_TARGET to 5 GB and INMEMORY_SIZE to 1 GB, then the overall memory target is 5 GB (not 6 GB), and the INMEMORY_SIZE is always 1 GB. The following graphic shows a database that sometimes processes jobs submitted by online users and sometimes batch jobs. Using automatic memory management, the database automatically adjusts the size of the large pool and database buffer cache depending on which type of jobs are running. Note that the graphic does not depict the IM column store or memoptimize area. If you create a database with DBCA and choose the basic installation option, then Oracle Database enables automatic memory management by default. Page 12Fast ingest optimizes the processing of high-frequency, single-row data inserts into database from applications, such as Internet of Things (IoT) applications.Fast ingest uses MEMOPTIMIZE_WRITE to insert data into tables specified as MEMOPTIMIZE FOR WRITE hint. The database temporarily buffers these inserts in the large pool and automatically commits the changes at the time of writing these buffered inserts to disk. The changes cannot be rolled back. The inserts using fast ingest are also known as deferred inserts, because they are initially buffered in the large pool and later written to disk asynchronously by background processes. Steps for using fast ingest for inserting data into a table The following are the steps for using fast ingest for inserting data into a table: Enable a table for fast ingest by specifying the MEMOPTIMIZE FOR WRITE hint in the CREATE TABLE or ALTER TABLE statement. SQL> create table test_fast_ingest (id number primary key, test_col varchar2(15)) memoptimize for write; Table created.See "Enabling a Table for Fast Ingest" for more information. Enable fast ingest for inserts by specifying the MEMOPTIMIZE_WRITE hint in the INSERT statement. The following is not how fast ingest is meant to be used, but demonstrates the mechanism.SQL> insert /*+ memoptimize_write */ into test_fast_ingest values (1, 'test'); 1 row created SQL> insert /*+ memoptimize_write */ into test_fast_ingest values (2, 'test'); 1 row createdSee "Specifying a Hint for Using Fast Ingest for Data Inserts" for more information. The result of the two inserts above is to write data to the ingest buffer in the large pool of the SGA. At some point, that data is flushed to the TEST_FAST_INGEST table. Until that happens, the data is not durable. Because the purpose of fast ingest is to support high performance data streaming, a more realistic architecture would involve having one or more application or ingest servers collecting data and batching inserts to the database. The first time an insert is run, the fast ingest area is allocated from the large pool. The amount of memory allocated is written to the alert.log. Details about fast ingest The intent of fast-ingest is to support applications that generate lots of informational data that has important value in the aggregate but that doesn't necessarily require full ACID requirements. Many applications in the Internet of Things (IoT) have a rapid "fire and forget" type workload, such as sensor data, smart meter data or even traffic cameras. For these applications, data might be collected and written to the database in high volumes for later analysis. The following diagram shows how this might work with the Memoptimized Rowstore – Fast Ingest feature. The ingested data is batched in the large pool and is not immediately written to the database. Thus, the ingest process is very fast. Very large volumes of data can be ingested efficiently without having to process individual rows. However, if the database goes down before the ingested data is written out to the database files, it is possible to lose data. Fast ingest is very different from normal Oracle Database transaction processing where data is logged and never lost once "written" to the database (i.e. committed). In order to achieve the maximum ingest throughput, the normal Oracle transaction mechanisms are bypassed, and it is the responsibility of the application to check to see that all data was indeed written to the database. Special APIs have been added that can be called to check if the data has been written to the database. The commit operation has no meaning in the context of fast ingest, because it is not a transaction in the traditional Oracle sense. There is no ability to rollback the inserts. You also cannot query the data until it has been flushed from the fast ingest buffers to disk. You can see some administrative information about the fast ingest buffers by querying the view V$MEMOPTIMIZE_WRITE_AREA. You can also use the packages DBMS_MEMOPTIMIZE and DBMS_MEMOPTIMIZE_ADMIN to perform functions like flushing fast ingest data from the large pool and determining the sequence id of data that has been written. Index operations and constraint checking is done only when the data is written from the fast ingest area in the large pool to disk. If primary key violations occur when the background processes write data to disk, then the database will not write those rows to the database. Assuming (for most applications but not all) that all inserted data needs to be written to the database, it is critical that the application insert process checks to see that the inserted data has actually been written to the database before destroying that data. Only when that confirmation has occurred can the data be deleted from the inserting process. Limitations for using fast ingest Tables with the following characteristics cannot use fast ingest: Tables with: disk compression in-memory compression column default vales encryption functional indexes domain indexes bitmap indexes bitmap join indexes ref types varray types OID$ types unused columns virtual columns LOBs triggers binary columns foreign keys row archival invisible columns Temporary tables Nested tables Index organized tables External tables Materialized views with on-demand refresh Sub-partitioning is not supported. The following partitioning types are not supported. REFERENCE SYSTEM INTERVAL AUTOLIST The following are some additional considerations for fast ingest: Because fast ingest buffers data in the large pool, there is a possibility of data loss in the event of a system failure. To avoid data loss, a client must keep a local copy of the data after performing inserts, so that it can replay the inserts in the event of a system failure before the data is written to disk. A client can use the DBMS_MEMOPTIMIZE package subprograms to track the durability of the inserts. After inserts are written to disk, a client can destroy its local copy of the inserted data. Queries do not read data from the large pool, hence data inserted using fast ingest cannot be queried until it is written to disk. Parent-child transactions must be synchronized to avoid errors. For example, foreign key inserts and updates of rows inserted into the large pool can return errors, if the parent data is not yet written to disk. Index operations are supported by fast ingest similar to the regular inserts. However, for fast ingest, database performs index operations while writing data to disk, and not while writing data into the large pool. JSON is only supported stored as a 4K VARCHAR2 and not as a LOB. Extended 32K string lengths are not supported (i.e. max_string_size=extended). The size allocated to the fast ingest buffers in the Large pool is fixed once created. If the buffer fills, further ingest waits until the background processes drain the buffer. Page 13The V$DB_CACHE_ADVICE view shows the simulated miss rates for a range of potential buffer cache sizes. This view assists in cache sizing by providing information that predicts the number of physical reads for each potential cache size. The data also includes a physical read factor, which is a factor by which the current number of physical reads is estimated to change if the buffer cache is resized to a given value. However, physical reads do not necessarily indicate disk reads in Oracle Database, because physical reads may be accomplished by reading from the file system cache. Hence, the relationship between successfully finding a block in the cache and the size of the cache is not always a smooth distribution. When sizing the buffer pool, avoid using additional buffers that do not contribute (or contribute very little) to the cache hit ratio.The following figure illustrates the relationship between physical I/O ratio and buffer cache size.There is some overhead associated with using this advisory view. When the advisory is enabled, there is a small increase in CPU usage, because additional bookkeeping is required. To reduce both the CPU and memory overhead associated with bookkeeping, Oracle Database uses sampling to gather cache advisory statistics. Sampling is not used if the number of buffers in a buffer pool is small to begin with. To use the V$DB_CACHE_ADVICE view: Set the value of the DB_CACHE_ADVICE initialization parameter to ON. This enables the advisory view. The DB_CACHE_ADVICE parameter is dynamic, so the advisory can be enabled and disabled dynamically to enable you to collect advisory data for a specific workload. Run a representative workload on the database instance. Allow the workload to stabilize before querying the V$DB_CACHE_ADVICE view. Query the V$DB_CACHE_ADVICE view. The following example shows a query of this view that returns the predicted I/O requirement for the default buffer pool for various cache sizes.COLUMN size_for_estimate FORMAT 999,999,999,999 heading 'Cache Size (MB)' COLUMN buffers_for_estimate FORMAT 999,999,999 heading 'Buffers' COLUMN estd_physical_read_factor FORMAT 999.90 heading 'Estd Phys|Read Factor' COLUMN estd_physical_reads FORMAT 999,999,999 heading 'Estd Phys| Reads' SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor, estd_physical_reads FROM V$DB_CACHE_ADVICE WHERE name = 'DEFAULT' AND block_size = (SELECT value FROM V$PARAMETER WHERE name = 'db_block_size') AND advice_status = 'ON'; The output of this query might look like the following: Estd Phys Estd Phys Cache Size (MB) Buffers Read Factor Reads ---------------- ------------ ----------- ------------ 30 3,802 18.70 192,317,943 10% of Current Size 60 7,604 12.83 131,949,536 91 11,406 7.38 75,865,861 121 15,208 4.97 51,111,658 152 19,010 3.64 37,460,786 182 22,812 2.50 25,668,196 212 26,614 1.74 17,850,847 243 30,416 1.33 13,720,149 273 34,218 1.13 11,583,180 304 38,020 1.00 10,282,475 Current Size 334 41,822 .93 9,515,878 364 45,624 .87 8,909,026 395 49,426 .83 8,495,039 424 53,228 .79 8,116,496 456 57,030 .76 7,824,764 486 60,832 .74 7,563,180 517 64,634 .71 7,311,729 547 68,436 .69 7,104,280 577 72,238 .67 6,895,122 608 76,040 .66 6,739,731 200% of Current Size In this example, the output shows that if the cache was 212 MB instead of the current size of 304 MB, the estimated number of physical reads would increase by a factor of 1.74, or 74%. Hence, it is not advisable to decrease the cache size to 212MB. However, increasing the cache size to 334MB may potentially decrease reads by a factor of .93, or 7%. If an additional 30MB memory is available on the system and the value of the SGA_MAX_SIZE parameter allows for the increment, it is advisable to increase the default buffer cache pool size to 334MB. Page 14 The library cache stores executable forms of SQL cursors, PL/SQL programs, and Java classes, which are collectively referred to as the application code. This section focuses on tuning as it relates to the application code. When the application code is executed, Oracle Database attempts to reuse existing code if it has been executed previously and can be shared. If the parsed representation of the SQL statement exists in the library cache and it can be shared, then the database reuses the existing code. This is known as a soft parse, or a library cache hit. If Oracle Database cannot use the existing code, then the database must build a new executable version of the application code. This is known as a hard parse, or a library cache miss. For information about when SQL and PL/SQL statements can be shared, see "SQL Sharing Criteria". In order to perform a hard parse, Oracle Database uses more resources than during a soft parse. Resources used for a soft parse include CPU and library cache latch gets. Resources required for a hard parse include additional CPU, library cache latch gets, and shared pool latch gets. A hard parse may occur on either the parse step or the execute step when processing a SQL statement. When an application makes a parse call for a SQL statement, if the parsed representation of the statement does not exist in the library cache, then Oracle Database parses the statement and stores the parsed form in the shared pool. To reduce library cache misses on parse calls, ensure that all sharable SQL statements are stored in the shared pool whenever possible. When an application makes an execute call for a SQL statement, if the executable portion of the SQL statement is aged out (or deallocated) from the library cache to make room for another statement, then Oracle Database implicitly reparses the statement to create a new shared SQL area for it, and executes the statement. This also results in a hard parse. To reduce library cache misses on execution calls, allocate more memory to the library cache. For more information about hard and soft parsing, see "SQL Execution Efficiency". Page 15 The RESULT_CACHE hint applies only to the query block in which the hint is specified. If the hint is specified only in a view, then only these results are cached. View caching has the following characteristics: The view must be one of the following types: A standard view (a view created with the CREATE ... VIEW statement) An inline view specified in the FROM clause of a SELECT statement An inline view created with the WITH clause The result of a view query with a correlated column (a reference to an outer query block) cannot be cached. Query results are stored in the server result cache, not the client result cache. A caching view is not merged into its outer (or referring) query block. Adding the RESULT_CACHE hint to inline views disables optimizations between the outer query and inline view to maximize reusability of the cached result. The following example shows a query of the inline view view1. SELECT * FROM (SELECT /*+ RESULT_CACHE */ department_id, manager_id, count(*) count FROM hr.employees GROUP BY department_id, manager_id) view1 WHERE department_id = 30; In this example, the SELECT statement from view1 is the outer block, whereas the SELECT statement from employees is the inner block. Because the RESULT_CACHE hint is specified only in the inner block, the results of the inner query are stored in the server result cache, but the results of the outer query are not cached. Assume that the same session run a query of the view view2 as shown in the following example. WITH view2 AS (SELECT /*+ RESULT_CACHE */ department_id, manager_id, count(*) count FROM hr.employees GROUP BY department_id, manager_id) SELECT * FROM view2 WHERE count BETWEEN 1 and 5; In this example, because the RESULT_CACHE hint is specified only in the query block in the WITH clause, the results of the employees query are eligible to be cached. Because these results are cached from the query in the first example, the SELECT statement in the WITH clause in the second example can retrieve the cached rows. Page 16 The V$PGA_TARGET_ADVICE view predicts how changing the value of the PGA_AGGREGATE_TARGET initialization parameter will affect the following statistics in the V$PGASTAT view: cache hit percentage over allocation count The following example shows a query of this view.SELECT ROUND(pga_target_for_estimate/1024/1024) target_mb, estd_pga_cache_hit_percentage cache_hit_perc, estd_overalloc_count FROM V$PGA_TARGET_ADVICE; The output of this query might look like the following: TARGET_MB CACHE_HIT_PERC ESTD_OVERALLOC_COUNT ---------- -------------- -------------------- 63 23 367 125 24 30 250 30 3 375 39 0 500 58 0 600 59 0 700 59 0 800 60 0 900 60 0 1000 61 0 1500 67 0 2000 76 0 3000 83 0 4000 85 0 The following figure illustrates how the result of this query can be plotted. The curve shows how PGA cache hit percentage improves as the value of the PGA_AGGREGATE_TARGET parameter increases. The shaded zone in the graph represents the over allocation zone, where the value of the ESTD_OVERALLOCATION_COUNT column is non-zero. This area indicates that the value of the PGA_AGGREGATE_TARGET parameter is too small to meet the minimum PGA memory requirements. If the value of the PGA_AGGREGATE_TARGET parameter is set within the over allocation zone, then the memory manager will over-allocate memory and the actual PGA memory consumed will exceed the limit that was set. It is therefore meaningless to set a value of the PGA_AGGREGATE_TARGET parameter in that zone. In this particular example, the PGA_AGGREGATE_TARGET parameter should be set to at least 375 MB. Beyond the over allocation zone, the value of the PGA cache hit percentage increases rapidly. This is due to an increase in the number of optimal or one-pass work areas and a decrease in the number of multi-pass executions. At some point, around 500 MB in this example, an inflection in the curve corresponds to the point where most (probably all) work areas can run in optimal or at least one-pass size. Beyond this point, the cache hit percentage keeps increasing, though at a lower pace, up to the point where it starts to taper off and only slight improvement is achieved with increase in the value of the PGA_AGGREGATE_TARGET parameter. In the figure, this happens when PGA_AGGREGATE_TARGET reaches 3 GB. At this point, the cache hit percentage is 83% and only marginal improvement (by 2%) is achieved with one extra gigabyte of PGA memory. In this example, 3 GB is the optimal value for the PGA_AGGREGATE_TARGET parameter. Ideally, the value of the PGA_AGGREGATE_TARGET parameter should be set to the optimal value, or at least to the maximum value possible in the region beyond the over allocation zone. As a rule of thumb, the PGA cache hit percentage should be higher than 60%, because at 60% the system is almost processing double the number of bytes it actually needs to process in an ideal situation. In this example, the value of the PGA_AGGREGATE_TARGET parameter should be set to at least 500 MB, and as close to 3 GB as possible. However, the correct setting for the PGA_AGGREGATE_TARGET parameter depends on how much memory can be dedicated to the PGA component. Generally, adding PGA memory requires reducing memory for some SGA components—like the shared pool or buffer cache—because the overall memory dedicated to the database instance is often bound by the amount of physical memory available on the system. Therefore, any decisions to increase PGA memory must be taken in the larger context of the available memory in the system and the performance of the various SGA components (which you can monitor with shared pool advisory and buffer cache advisory statistics). If you cannot reduce memory from the SGA components, consider adding more physical memory to the system. Page 17 Page 18 mytest_summary.txt: This file contains: Input parameters Maximum throughput observed for the Large Random/Sequential workload Maximum I/O rate observed for the Small Random workload Minimum latency observed for the Small Random workload mytest_mbps.csv: comma-delimited value file containing the data transfer rate (MBPS) results for the Large Random/Sequential workload. In the general case, this and all other CSV files contains a two-dimensional table. Each row in the table corresponds to a large I/O load level and each column corresponds to a specific small I/O load level. Thus, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (for random large I/O tests) or the number of sequential streams (for sequential large I/O tests). The following example shows the first few data points of the Orion MBPS output CSV file for "mytest". The simple mytest command-line does not test combinations of large and small I/Os. Hence, the MBPS file has just one column corresponding to 0 outstanding small I/Os. In this example, at a load level of 8 outstanding large reads and no small I/Os, the report data indicates a throughput of 103.06 MBPS.Large/Small, 0 1, 19.18 2, 37.59 4, 65.53 6, 87.03 8, 103.06 10, 109.67 . . . . .The following graph shows a sample data transfer rate measured at different large I/O load levels. This graph can be generated by loading mytest_mbps.csv into a spreadsheet and graphing the data points. Orion does not directly generate such graphs. The x-axis corresponds to the number of outstanding large reads and the y-axis corresponds to the throughput observed. The graph shows typical storage system behavior. As the number of outstanding I/O requests is increased, the throughput increases. However, at a certain point the throughput level stabilizes, indicating the storage system's maximum throughput value. mytest_iops.csv: Comma-delimited value file containing the I/O throughput (in IOPS) results for the Small Random workload. Like in the MBPS file, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os, when testing large random, or the number of sequential streams (for large sequential). In the general case, a CSV file contains a two-dimensional table. However, for a simple test where you are not testing combinations of large and small I/Os the results file has just one row. Hence, the IOPS results file just has one row with 0 large I/Os. As shown in the following example, data point with 12 outstanding small reads and no large I/Os provides a sample throughput of 951 IOPS.Large/Small, 1, 2, 3, 6, 9, 12 . . . . 0, 105, 208, 309, 569, 782, 951 . . . . The following graph is generated by loading mytest_iops.csv into Excel and charting the data. This graph illustrates the IOPS throughput seen at different small I/O load levels. The graph shows typical storage system behavior. As the number of outstanding I/O requests is increased, the throughput increases. However, at a certain point, the throughput level stabilizes, indicating the storage system reaches a maximum throughput value. At higher throughput levels, the latency for the I/O requests also increase significantly. Therefore, it is important to view this data with the latency data provided in the generated latency results in mytest_lat.csv. mytest_lat.csv: Comma-delimited value file containing the latency results for the Small Random workload. As with the MBPS and IOPS files, the column headings are the number of outstanding small I/Os and the row headings are the number of outstanding large I/Os (when testing large random I/Os) or the number of sequential streams. In the general case, a CSV file contains a two-dimensional table. However, for a simple test where you are not testing combinations of large and small I/Os the results file has just one row. Hence, the IOPS results file just has one row with 0 large I/Os. In the following example, at a sustained load level of 12 outstanding small reads and no large I/Os, the generated results show an I/O turn-around latency of 22.25 milliseconds.Large/Small, 1, 2, 3, 6, 9, 12 . . . . 0, 14.22, 14.69, 15.09, 16.98, 18.91, 21.25 . . . . The following graph is generated by loading mytest_lat.csv into Excel and charting the data. This graph illustrates the small I/O latency at different small I/O load levels for mytest. mytest_trace.txt: Contains the extended, unprocessed test output. Page 19 To address CPU problems, first establish appropriate expectations for the amount of CPU resources your system should be using. Then, determine whether sufficient CPU resources are available and recognize when your system is consuming too many resources. Begin by determining the amount of CPU resources the Oracle database instance utilizes with your system in the following three cases: System is idle, when little Oracle Database and non-Oracle activity exists System at average workloads System at peak workloads You can capture various workload snapshots using the Automatic Workload Repository, Statspack, or the UTLBSTAT/UTLESTAT utility. Operating system utilities—such as vmstat, sar, and iostat on UNIX and the administrative performance monitoring tool on Windows—can be used along with the V$OSSTAT or V$SYSMETRIC_HISTORY view during the same time interval as Automatic Workload Repository, Statspack, or UTLBSTAT/UTLESTAT to provide a complimentary view of the overall statistics. Workload is an important factor when evaluating your system's level of CPU utilization. During peak workload hours, 90% CPU utilization with 10% idle and waiting time can be acceptable. Even 30% utilization at a time of low workload can be understandable. However, if your system shows high utilization at normal workload, then there is no room for a peak workload. For example, The following figure illustrates workload over time for an application having peak periods at 10:00 AM and 2:00 PM. This example application has 100 users working 8 hours a day. Each user entering one transaction every 5 minutes translates into 9,600 transactions daily. Over an 8-hour period, the system must support 1,200 transactions an hour, which is an average of 20 transactions a minute. If the demand rate were constant, then you could build a system to meet this average workload. However, usage patterns are not constant and in this context, 20 transactions a minute can be understood as merely a minimum requirement. If the peak rate you need to achieve is 120 transactions a minute, then you must configure a system that can support this peak workload. For this example, assume that at peak workload, Oracle Database uses 90% of the CPU resource. For a period of average workload, then, Oracle Database uses no more than about 15% of the available CPU resource, as illustrated in the following equation: 20 tpm / 120 tpm * 90% = 15% of available CPU resourcewhere tpm is transactions a minute. If the system requires 50% of the CPU resource to achieve 20 tpm, then a problem exists: the system cannot achieve 120 transactions a minute using 90% of the CPU. However, if you tuned this system so that it achieves 20 tpm using only 15% of the CPU, then, assuming linear scalability, the system might achieve 120 transactions a minute using 90% of the CPU resources. As users are added to an application, the workload can rise to what had previously been peak levels. No further CPU capacity is then available for the new peak rate, which is actually higher than the previous. Page 20 This package contains procedures for controlling automatic materialized views. DBMS_AUTO_MV.CONFIGURE The DBA can use the CONFIGURE procedure of DBMS_AUTO_MV to start, stop, and configure automatic materialized views. Table 19-1 Configure Procedure Parameters Parameter Description and Examples AUTO_MV_MODE IMPLEMENT: Implements automatic materialized views. OFF: Turns off automatic materialized views. REPORT ONLY: Report-only mode. exec dbms_auto_mv.configure('AUTO_MV_MODE', 'IMPLEMENT'); exec dbms_auto_mv.configure('AUTO_MV_MODE', 'OFF'); exec dbms_auto_mv.configure('AUTO_MV_MODE', 'REPORT ONLY'); AUTO_MV_MAINT_TASK ENABLE: Activates the task performing the maintenance (refreshes, validations, and cleanup). DISABLE: Deactivates the task performing the maintenance. CLEANUP_AND_DISABLE: Drops all automatic materialized views and deactivates the task. If automatic materialized views maintenance is in progress, then maintenance is allowed to finish before the task is deactivated. exec dbms_auto_mv.configure('AUTO_MV_MAINT_TASK', 'ENABLE'); exec dbms_auto_mv.configure('AUTO_MV_MAINT_TASK', 'DISABLE'); exec dbms_auto_mv.configure('AUTO_MV_MAINT_TASK', 'CLEANUP_AND_DISABLE'); AUTO_MV_SPACE_BUDGET Specifies the percentage of space budgeted for implementing automatic materialized views within the tablespace where those views were created. This is a percentage of the total space used by all automatic materialized views and associated indexes within the tablespace. A condition on the enforcement of AUTO_MV_SPACE_BUDGET is the value of AUTO_MV_DEFAULT_TABLESPACE: If AUTO_MV_DEFAULT_TABLESPACE is not defined (NULL), then automatic materialized views are created on the tablespace of the view's parent object (which is the largest FACT table in the view's definition). In this case, the budget defined by AUTO_MV_SPACE_BUDGET is enforced within that tablespace. If AUTO_MV_DEFAULT_TABLESPACE is defined, then automatic materialized views are created in the designated default tablespace. In this case, the budget set by AUTO_MV_SPACE_BUDGET is ignored. If the budget is exceeded (possibly because of the growth of automatic materialized views), then the least-used automatic materialized view is dropped. The value is an integer from 1 to 100. The default is 67 (67% of the total volume of the tablespace). exec dbms_auto_mv.configure('AUTO_MV_SPACE_BUDGET', 15); AUTO_MV_DEFAULT_TABLESPACE Specifies the default tablespace for the creation of automatic

materialized views. Possible values are the name of a valid temporary tablespace or NULL (the default). In the case of NULL, new automatic materialized view is created in the default tablespace of the owner of the parent object. If the view has more than one parent object, such as materialized views defined on multiple base tables, then the default tablespace of the owner of largest base table is selected. If the value is changed dynamically, the change takes effect the next time automatic materialized view recommendations are implemented. exec dbms_auto_mv.configure('AUTO_MV_DEFAULT_TABLESPACE','MYTABLESPACE'); exec dbms_auto_mv.configure('AUTO_MV_DEFAULT_TABLESPACE'); AUTO_MV_TEMP_TABLESPACE Specifies the temporary tablespace used for creation or refresh of automatic materialized views. Possible values are the name of a valid temporary tablespace or NULL. In the case of NULL, the tablespace is assigned to the owner of the largest parent object of the automatic materialized views. The default is NULL.exec dbms_auto_mv.configure('AUTO_MV_TEMP_TABLESPACE','TEMP2'); exec dbms_auto_mv.configure('AUTO_MV_TEMP_TABLESPACE'); AUTO_MV_RETENTION Specifies the number of days automatic materialized view can continue to exist without being queried. If an automatic materialized view remains unused beyond this retention time, it is automatically dropped. Possible values are any integer between 1 and 373. The default is 33 days. exec dbms_auto_mv.configure('AUTO_MV_RETENTION', 365); AUTO_MV_ANALYZE_REPORT_RETENTION AUTO_MV_ANALYZE_REPORT_RETENTION Specifies the maximum number of days to retain analysis and recommendation history. Possible values are any integer from 0 to 90. A value of 0 means no history is maintained. The default is 31 days. exec dbms_auto_mv.configure('AUTO_MV_ANALYZE_REPORT_RETENTION', 60); AUTO_MV_VERIFY_REPORT_RETENTION Specifies the maximum number of days to retain verification history. Possible values are any integer from 0 to 90. The value 0 specifies that no verification history will be maintained. The default is 31 days. exec dbms_auto_mv.configure('AUTO_MV_VERIFY_REPORT_RETENTION', 7); AUTO_MV_MAINT_REPORT_RETENTION Specifies the maximum number of days to retain history of automatic materialized view maintenance (refreshes) in the DBA_AUTO_MV_REFRESH_* dictionary tables. Possible values are any integer from 0 to 90. The value 0 specifies that no refresh history will be maintained. The default is 31 days.exec dbms_auto_mv.configure('AUTO_MV_MAINT_REPORT_RETENTION', 14); AUTO_MV_ANALYZE_WORKLOAD_WINDOW Specifies the maximum number of hours to investigate queries from the latest snapshots and make recommendations. Possible values are any integer between from 1 to 8760. The default is 24 hours.exec dbms_auto_mv.configure('AUTO_MV_ANALYZE_WORKLOAD_WINDOW', 48); AUTO_MV_ANALYZE_WORKLOAD_MIN_TIME Specifies the minimum time in seconds for a query to be considered for automatic materialized views recommendation. Queries below this threshold are not considered for recommendations. Possible values are any integer from 0 to 3600. The default is 120 seconds. exec dbms_auto_mv.configure('AUTO_MV_ANALYZE_WORKLOAD_MIN_TIME', 1800); AUTO_MV_SCHEMA Specifies the schema to be either included or excluded during the creation of automatic materialized views. The schema is added to the inclusion list or the exclusion list in the configuration. Initially, both lists are empty and automatic materialized views can be created in all the schemas in a database where automatic materialized views are enabled. You can build the inclusion and exclusion lists by calling AUTO_MV_SCHEMA multiple times. The boolean ALLOW determines if the schema is added to the inclusion list (TRUE) or to the exclusion list (FALSE). The default is TRUE. During workload processing, any query that does not contain a reference at least one table in a schema on the inclusion list is not analyzed and not auto tuned. It is not factored into recommendations and verifications. Likewise, if a query references a table in a schema on the exclusion list, that query is excluded from processing. exec dbms_auto_mv.configure('AUTO_MV_SCHEMA', 'SCHEMA_A'); exec dbms_auto_mv.configure('AUTO_MV_SCHEMA', 'SCHEMA_B', FALSE);To enable or disable processing of all schemas, you can specific the schema as NULL. This either enables or disables all of them, depending on the value of ALLOW. exec dbms_auto_mv.configure('AUTO_MV_SCHEMA','', TRUE); AUTO_MV_APP_MODULE Specifies application modules to include or exclude from the creation of automatic materialized views.exec dbms_auto_mv.configure('AUTO_MV_APP_MODULE', 'MODULE1', TRUE); exec dbms_auto_mv.configure('AUTO_MV_APP_MODULE', 'MODULE1', FALSE); exec dbms_auto_mv.configure('AUTO_MV_APP_MODULE', 'MODULE1'); DBMS_AUTO_MV.DROP_AUTO_MVS This procedure drops an automatic materialized view. It can be executed only by users who have the DBA role. Parameter Description OWNER The name of the owner of the automatic materialized view. MV_NAME The name of the automatic materialized view. ALLOW_RECREATE Allow the materialized view to be recreated if necessary. Optional. Note that if OWNER is specified and MV_NAME is set to NULL, then all automatic materialized views owned by OWNER are dropped. exec dbms_auto_mv.drop_auto_mvs('SH', 'AUTO_MV$$_G2MKPB9SA1FB7', TRUE); exec dbms_auto_mvs('SH', 'AUTO_MV$$_G2MKPB9SA1FB7'); exec dbms_auto_mv.drop_auto_mvs('SH', '', TRUE); exec dbms_auto_mv.drop_auto_mvs('SH', ''); DBMS_AUTO_MV.RECOMMEND DBMS_AUTO_MV.RECOMMEND generates automatic materialized recommendations based on a given SQL tuning set. This API enables you to manually run automatic materialized view analysis and verification from a command line (instead of through an Automatic SQL Tuning task). You set the workload start and end time and determine whether this execution results in a report only, or an actual implementation. There is no default time limit for the workload window. Execution of this API requires the DBA role. Parameter Description STS_OWNER The name of the owner of the SQL tuning set. Default: SYS. STS_NAME The name of the SQL tuning set. Default: SYS_AUTO_STS. WORKLOAD_START_TIME Start time for the workload window. Default: WORKLOAD_END_TIME minus 24 hours. WORKLOAD_END_TIME End time for the workload window. Default: The current timestamp. AUTO_MV_MODE REPORT ONLY (recommendations only) or IMPLEMENT. Default: REPORT ONLY Example: Generate and report recommendations using SYS_AUTO_STS for the past 24 hours. Note that the default behavior is REPORT_ONLY, which means that no automatic materialized view will be implemented. var exec_name varchar2(200); begin :exec_name := dbms_auto_mv.recommend(); end; SELECT * FROM DBA_AUTO_MV_ANALYSIS_RECOMMENDATIONS WHERE exec_name = :exec_name; DBMS_AUTO_MV.REFRESH The DBMS_AUTO_MV.RECOMMEND API enables you force a refresh of all stale automatic materialized views. The stale automatic materialized views are unconditionally refreshed in descending order, based on their verified query rewrite benefit values. There are no parameters. This routine can be executed only by users with the DBA role. exec dbms_auto_mv.dbms_auto_refresh(); DBMS_AUTO_MV.REPORT_ACTIVITY The DBMS_AUTO_MV.REPORT_ACTIVITY This API generates a report on automatic materialized view activities and usage within a specified time window. The report is returned as a CLOB. Parameter Description ACTIVITY_START The start of the time window. Default: SYSTIMESTAMP -1. ACTIVITY_END The end of the time window. Default: SYSTIMESTAMP. TYPE The format of the report. 'TEXT', 'HTML', and 'XML' are supported. Default: 'TEXT'. SECTION The section or sections covered by the report. The value can be any combination of: SUMMARY, MV_DETAILS, QUERY_DETAILS, VERIFICATION_DETAILS or ALL. Default: 'ALL'. LEVEL The level of detail in the report: BASIC, TYPICAL or ALL. Default: 'TYPICAL'. Examples: Generate a report on all automatic materialized view activities. Output the report in HTML format: select dbms_auto_mv.report_activity(type => 'HTML') from dual; Generate a report on all automatic materialized view activities. Exclude the verification details. Output the report in XML format. select dbms_auto_mv.report_activity(type => 'XML', section => 'ALL-VERIFICATION_DETAILS') from dual; DBMS_AUTO_MV.REPORT_LAST_ACTIVITY The DBMS_AUTO_MV.REPORT_LAST_ACTIVITY API generates a report on the most recent automatic materialized view activities and usage. Parameter Description TYPE The format of the report. 'TEXT', 'HTML', and 'XML' are supported. Default: 'TEXT'. SECTION The section or sections covered by the report. The value can be any combination of: SUMMARY, MV_DETAILS, QUERY_DETAILS, VERIFICATION_DETAILS or ALL. Default: 'ALL'. LEVEL The level of detail in the report: BASIC, TYPICAL or ALL. Default: 'TYPICAL'. Examples: Generate a comprehensive report of the most recent activity, at the typical level of detail. Output the report in text format (the default). Note that both of these statements return the same results. select dbms_auto_mv.report_last_activity('TEXT', 'ALL', 'TYPICAL') from dual; select dbms_auto_mv.report_last_activity() from dual; Generate a report of the most recent activity that includes only the summary and the details of the materialized view. Report at the maximum level of detail. Output in XML format. select dbms_auto_mv.report_last_activity('XML', 'SUMMARY+MV_DETAILS', 'ALL') from dual; Generate a report of the most recent activity at the basic level of detail. Exclude the verification details. Output in HTML format. select dbms_auto_mv.report_last_activity('XML', 'ALL-VERIFICATION_DETAIL', 'BASIC') from dual; Page 21 A policy that specifies a rule and condition for Automatic Data Optimization (ADO). For example, an ADO policy may specify that an object is marked NOINMEMORY (action) 30 days after creation (condition). Specify ADO policies using the ILM clause of CREATE TABLE and ALTER TABLE statements. A feature that automatically evicts cold (infrequently accessed) segments from the IM column store to ensure that the working data set is always populated. The degree to which an application, service, or function is accessible on demand. A low-memory data structure that tests membership in a set. The database uses Bloom filters to improve the performance of hash joins. Contiguous storage for a column in an In-Memory Compression Unit (IMCU). The subpool in the In-Memory Area that stores columnar data. It is also known as the 1 MB pool. The columnar format used in IMCUs. The columnar format used in the IM column store. The columnar format contrasts with the row format used in data blocks. A segment-level, instance-specific set of master dictionary codes, created from local dictionaries. A local dictionary is a sorted list of dictionary codes specific to a Column Compression Unit (CU). A join group uses a common dictionary to optimize joins. The application of different levels of compression to data based on its access pattern. For example, administrators may compress inactive data at a higher rate of compression at the cost of slower access. The unit of work between data redistribution stages in a parallel query. A key that represents all grouping keys whose grouping columns come from a specific fact table or dimension. A key that represents all join keys whose join columns come from a particular fact table or dimension. A numeric key that is stored as a native integer and has a range of values. A repopulation mechanism in which background processes create new In-Memory Compression Unit (IMCU) versions by combining the original rows with the latest modified rows. During repopulation, the stale IMCUs remain accessible for queries. A combination of one or more values, operators, and SQL functions that resolves to a value. The time interval within which the database considers IM expressions for possible capture. An expression capture interval defined by invocation of the IME_OPEN_CAPTURE_WINDOW and IME_CLOSE_CAPTURE_WINDOW procedures in the DBMS_INMEMORY_ADMIN package. A repository maintained by the optimizer to store statistics about expression evaluation. For each segment, the ESS monitors statistics such as frequency of execution, cost of evaluation, timestamp evaluation, and so on. The ESS is persistent in nature and has an SGA representation for fast lookup of expressions. Heat Map shows the popularity of data blocks and rows. Automatic Data Optimization (ADO) to decide which segments are candidates for movement to a different storage tier. The database instance in which an IMCU resides. When auto DOP is enabled on Oracle RAC, the parallel query coordinator uses home location to determine where each IMCU is located, how large it is, and so on. A query that scans both the IM column store and the row store. The optimizer considers an In-Memory hybrid scan automatically when all predicate columns have the INMEMORY attribute, and some columns in the SELECT list do not have the INMEMORY attribute. A table in which some partitions are stored in data file segments and some are stored in external data source. An optimization that accelerates aggregation for queries that join from a single large table to multiple small tables. The transformation uses KEY VECTOR and VECTOR GROUP BY operators, which is why it is also known as VECTOR GROUP BY aggregation. An optional SGA area that stores copies of tables and partitions in a columnar format optimized for rapid scans. The use of lightweight threads to automatically parallelize In-Memory table scans. A SQL expression whose results are stored in the In-Memory Column Store. If last_name is a column stored in the IM column store, then an IM expression might be UPPER(last_name). In Oracle RAC, the duplication of an IMCU in multiple IM column stores. For example, the IM column stores on instance 1 and instance 2 are populated with the same sales table. In a query of the In-Memory Column Store, the elimination of IMCUs based on the high and low values in each IMCU. For example, if a statements filters product IDs greater than 100, then the database avoids scanning IMCUs that contain values less than 100. A data structure in an IMCU header that stores the minimum and maximum for all columns within the IMCU. A downloadable PL/SQL package that analyzes the analytical processing workload in your database. This advisor recommends a size for the IM column store and a list of objects that would benefit from In-Memory population. An optional SGA component that contains the IM column store. A storage unit in the In-Memory Column Store that is optimized for faster scans. The In-Memory Column Store stores each column in table separately and compresses it. Each IMCU contains all columns for a subset of rows in a specific table segment. A one-to-many mapping exists between an IMCU and a set of database blocks. For example, if a table contains columns c1 and c2, and if its rows are stored in 100 database blocks on disk, then IMCU 1 might store the values for both columns for blocks 1-50, and IMCU 2 might store the values for both columns for blocks 51-100. A background process whose primary task is to initiate background population and repopulation of columnar data. A feature that significantly reduces the time to populate data into the IM column store when a database instance restarts. A set of processes and policies for managing data throughout its useful life. A user-defined object that specifies frequently joined columns from the same table or different tables. External tables are not supported. A typical join group candidate is a set of columns used to join fact and dimension tables. Join groups are only supported when INMEMORY_SIZE is a nonzero value. A data structure that maps between dense join keys and dense grouping keys. Optional area in the SGA that provides large memory allocations for backup and restore operations, I/O server processes, and session memory for the shared server and Oracle XA. An execution entity used in an In-Memory Dynamic Scan. Lightweight threads help to parallelize scans of IMCUs. A subpool of the In-Memory Area that stores metadata about the objects that reside in the IM column store. The metadata pool is also known as the 64 KB pool. An SGA pool that stores buffers and related structures for heap-organized tables specified as MEMOPTIMIZE FOR READ. When INMEMORY PRIORITY is set to NONE, the IM column store only populates the object when it is accessed through a full scan. If the object is never accessed, or if it is accessed only through an index scan or fetch by rowid, then it is never populated. OSON enables fast queries and updates of the JSON data model in Oracle database server and Oracle database clients. A proprietary compression technique that offers extremely fast decompression. OZIP is tuned specifically for Oracle Database. A technique in which you create a table, load data into it, and then exchange an existing table partition with the table. This exchange process is a DDL operation with no actual data movement. The operation of reading existing data blocks from data files, transforming the rows into columnar format, and then writing the columnar data to the IM column store. In contrast, loading refers to bringing new data into the database using DML or DDL. When PRIORITY is set to a value other than NONE, Oracle Database adds the object to a prioritized population queue. The database populates objects based on their queue position, from CRITICAL to LOW. It is "priority-based" because the IM column store automatically populates objects using the prioritized list whenever the database re-opens. Unlike in on-demand population, objects do not require a full scan to be populated. A database instance that can process DML and supports direct client connections. By default, a database instance is read/write. The logical representation of an application workload that shares common attributes, performance thresholds, and priorities. A single service can be associated with one or more instances of an Oracle RAC database, and a single instance can support multiple services. System global area. A group of shared memory structures that contain data and control information for one Oracle database instance. Single Instruction, Multiple Data. An instruction that processes data as a single unit, called a vector, rather than as separate instructions. SIMD processing is known as vectorization. The deployment of data on different tiers of storage depending on its level of access. For example, administrators migrate inactive data from high-performance, high-cost storage to low-cost storage. A foreground or PQ process that coordinates with an IM dynamic scan. A column that is not stored on disk. The database derives the values in virtual columns on demand by computing a set of expressions or functions. The subset of INMEMORY objects that is actively queried at a given time. Typically, the work working data set changes over time. Page 22 Previous Next JavaScript must be enabled to correctly display this content

A B C D E F H I L M N O P Q R S T U V W Active Session History ADDM enabling in a PDB 7.1.3.1 allocation of memory 11.1 applications Automatic Database Diagnostic Monitor actions and rationales of recommendations 7.1.5 analysis results example 7.1.6 and DB time 7.1.1 example report 7.1.6 findings 7.1.5 results 7.1.5 setups 7.2 types of problems considered 7.1.1 types of recommendations 7.1.5 automatic database diagnostic monitoring 1.2.1 automatic memory management 11.2.2 automatic segment-space management 4.1.4, 10.3.2, 17.2.6.2 Automatic shared memory management 12.1 automatic SQL tuning 1.2.1 automatic undo management 4.1.2 Automatic Workload Repository 1.2.1 Active Data Guard support 6.2.8 AWR data storage and retrieval in a multitenant environment 6.2.7.2 categorization of AWR statistics in a multitenant environment 6.2.7.1 compare periods report configuring 6.2.1 default settings 6.1.4 factors affecting space usage 6.1.4 managing snapshots in ADG standby databases 6.2.8.2 minimizing space usage 6.1.4 modifying snapshot settings 6.2.2.4 multitenant environment support 6.2.7 overview 6.1 recommendations for retention period 6.1.4 reports 6.3.2.1 retention period 6.1.4 space usage 6.1.4 statistics collected 6.1 turning off automatic snapshot collection 6.1.4 unusual percentages in reports 6.3 Viewing AWR data in a multitenant environment 6.2.7.3 viewing remote snapshots for ADG standby databases 6.2.8.3 views for accessing data 6.2.6 Autonomous Data Warehouse Autonomous Transaction Processing awrrpt.sql Automatic Workload Repository report 6.3.2.1 data database caching modes configuring 13.5 default database caching mode 13.5.1 determining which mode to use 13.5.3 force full database caching mode verifying 13.5.4 database monitoring 1.2.1 database performance comparing 8.1 Database Resource Manager 10.1.2.1.3, 18.1.3, 18.4.2 databases diagnosing and monitoring 7.1 size 2.4.2 statistics 5.1 database time comparing over time 8.1 transient performance problems 9.1 DB_BLOCK_SIZE initialization parameter 17.2.1.2 DB_DOMAIN initialization parameter 4.1.1 DB_NAME initialization parameter 4.1.1 DBA_OBJECTS view 13.5.5.2 db block gets from cache statistic 13.2.2 db file scattered read wait events 10.3.3 db file sequential read wait events 10.3.3, 10.3.4 DBMS_ADVISOR package setting DBIO_EXPECTED 7.2 setups for ADDM 7.2 DBMS_SHARED_POOL package managing the shared pool 14.3.5 DB time metric 7.1.1 statistic 5.1.1 debugging designs 2.6.4 deploying applications 2.7 design principles 2.5 designs debugging 2.6.4 testing 2.6.4 validating 2.6.4 development environments 2.5.6 diagnostic monitoring 1.2.1, 7.1 dictionary cache 8.3.2.14 direct path disks monitoring operating system file activity 10.1.2.2 FAST_START_MTTR_TARGET and tuning instance recovery 10.4.3 fast lookup disabling for a table 12.6.3.3 Fast-Start checkpointing architecture 10.4.2 Fast-Start Fault Recovery 10.4, 10.4.2 free buffer wait events 10.3.9 free lists 10.3.2 function-based indexes 2.5.3.2 object-orientation 2.5.7 OPEN_CURSORS initialization parameter 4.1.1 operating system data cache 18.1.1 monitoring disk I/O 10.1.2.2 optimization optimizer introduction 1.1.3.1 query 1.1.3.1 Oracle CPU statistics 10.1.2.1.3 Oracle Enterprise Manager Cloud Control 1.2 advisors 1.2.1 Performance page 1.2.1 Oracle Forms control of parsing and private SQL areas 14.2.6.5 Oracle Managed Files 17.2.5 Oracle Orion calibration tool parameters 17.4.4 command-line options 17.4.4 Oracle performance improvement method 3.1 page table 18.1.4.2 paging 18.4.1.2 PARALLEL clause CREATE INDEX statement 4.2.3 parameter RESULT_CACHE_MAX_TEMP_RESULT 15.2.3 RESULT_CACHE_MAX_TEMP_SIZE 15.2.3 RESULT_CACHE_MODE 15.2.3 parameters parsing partitioned indexes 2.5.3.2 performance emergencies 3.2 improvement method 3.1 improvement method steps 3.1.1 mainframe 18.2.3 monitoring memory on Windows 18.4.1.1 tools for diagnosing and tuning 1.2 tools for UNIX-based systems 18.2.1 Windows 18.2.2 Performance Hub active reports about 6.4.1 generating 6.4, 6.4.3 performance problems transient performance tuning Fast-Start Fault Recovery 10.4 instance recovery 10.4 FAST_START_MTTR_TARGET 10.4.3 setting FAST_START_MTTR_TARGET 10.4.2 using V$INSTANCE_RECOVERY 10.4.2.3 per-session PGA memory limit PGA_AGGREGATE_TARGET initialization parameter 4.1.1 physical reads from cache statistic 13.2.2 proactive monitoring 1.1.2.4.1 processes Program Global Area program global area (PGA) direct path read 10.3.5 direct path write 10.3.6 shared servers 14.4.1 programming languages 2.5.6 queries query optimizer 1.1.3.1 Page 23 Proactive monitoring usually occurs on a regularly scheduled interval, where several performance statistics are examined to identify whether the system behavior and resource usage has changed. Proactive monitoring can also be considered as proactive tuning. Usually, monitoring does not result in configuration changes to the system, unless the monitoring exposes a serious problem that is developing. In some situations, experienced performance engineers can identify potential problems through statistics alone, although accompanying performance degradation is usual. Experimenting with or tweaking a system when there is no apparent performance degradation as a proactive action can be a dangerous activity, resulting in unnecessary performance drops. Tweaking a system should be considered reactive tuning, and the steps for reactive tuning should be followed. Monitoring is usually part of a larger capacity planning exercise, where resource consumption is examined to see changes in the way the application is being used, and the way the application is using the database and host resources. Oracle performance tuning is a crucial step in ensuring speedy application function and data retrieval. Here's what you need to know to improve database performance. All database administrators (DBAs) are familiar with the onerous task of increasing database performance. To accelerate application function, DBAs have to expedite query response time, which means DBAs must have a clear understanding of how their database is organized and how it serves its purpose. That is to say, DBAs must understand not just the database itself but the dedicated computer language that accesses the database to retrieve, manipulate, or delete information. What is Oracle Performance Tuning What is Relational Database? How does Oracle Performance Tuning work? Steps for effective Oracle Tuning Software recommendations Wondering how to do performance tuning in Oracle, specifically? Oracle is a relational database management system (RDBMS), and it utilizes Structured Query Language (SQL) to enable communication between applications and the database. Performance tuning is the process of optimizing Oracle performance by streamlining the execution of SQL statements. In other words, performance tuning simplifies the process of accessing and altering information contained by the database with the intention of improving query response times and application operations. To break this down further, let's first analyze the different components at play in a database management system, beginning with the database itself. As mentioned above, Oracle utilizes the relational database model. A relational database is an information system. That is, it's a computerized method of storing and using information. A relational database exists to consolidate, hold, and retrieve information as needed by applications. Those application will then use that data for some purpose. A company, for example, might use a database to store customers' personal information. A segmentation app may query the database for the email addresses of buyers who have made a purchase in the last month — personal information the company can use to target a specific customer segment with an email blast. A relational database stores data in tables. These tables are called "relations" and are configured much like a spreadsheet: the columns are "fields" that contain different attributes, and the rows are specific entries. A client base information table might have rows 1 to 27, each row corresponding to a different customer record and each column designating an attribute of personal information (name, email address, date of last purchase, etc.). Row 1 might read: "John," "john@email.com," "1/1/2019," etc. This database model is the foundation of the management system. The management system itself is software that creates relational databases and manages their organization and interaction with applications that use its data. As outlined by Oracle itself, the defining features of a database management system (DBMS) are: Kernel code – determines settings and allocates memory and storage for the system. A data dictionary – a collection of metadata. This repository offers a read-only overview of the data within the database, showing tables and views in addition to reference information regarding the database itself and its users. A specified query language – enables applications to access the database. As stated, Oracle uses SQL as its database language. SQL is distinguished by the ability to input, manipulate, retrieve, and delete data in a database. It also allows administrators to assign and revoke access as well as create their own views and functions. SQL is nonprocedural, which means that SQL statements inform the DBMS of what needs to be done, but they do not prescribe a course of action. Basically, the software in the DBMS figures out how to execute a statementby analyzing available options and electing the best sequence of actions. Not only is SQL a quick language to pick up, but users can embed SQL within other host languages. Performance tuning considers the many elements in an RDBMS to troubleshoot the source of performance problems. Often, DBAs are faced with a difficult task—network users report experiencing app delays and slow-loading pages, but administrators cannot pinpoint the source of database bottlenecks. Is it an optimizer issue? The coding of query statements? A problem with the computer itself? To identify the root cause, performance-tuning administrators must consider the many elements in an RDBMS to troubleshoot database operation lags. Not only can performance tuning be quite time-intensive, but it can be difficult to know where to begin. An intensive performance tuning process takes a systems-level approach and considers RDBMS components from top to bottom. As many database experts note, tuning SQL statements one by one will have little effect if the administrator hasn't first performed system-level tuning on the server, instance(s), and objects. I recommend assessing input and output (I/O) measures, optimizer parameters and statistics, and instance settings before tuning individual SQL statements. Otherwise, meticulous SQL tuning may be reversed later by the optimizer as it determines execution protocol contrary to designed execution plans. Once DBAs have done a systems-level check, they can proceed to SQL query tuning. Generally speaking, SQL tuning seeks to minimize the number of steps—"database touches"—a query entails, thus decreasing time cost and wait time. There are a lot of little SQL quirks and best practices to keep in mind, and while this list is by no means comprehensive or universally applicable, I have found that the following guidelines are helpful pointers in most situations. Here are my top 10 SQL query performance tuning tips: Begin by identifying the most cost-intensive queries to appropriately allocate your tuning efforts. The truth is that SQL query performance tuning is an ongoing process; there's always room for improvement, there's always more code to optimize, and there's always monitoring and upkeep to be done—which can make it feel like it never ends. For this reason, it's important to isolate the high-impact SQL statements—those that are executed most frequently and require the most database and I/O activity. These statements offer the biggest returns in terms of database performance improvement, so targeting them will optimize the amount of tuning work put in relative to performance improvements. Always minimize the amount of data that must be scanned in an operation. Many query statements will prompt the database to perform full-table scans, which incur much more I/O and can degrade performance by slowing down operations and carrying out unnecessarily broad searches. To streamline data retrieval: Add indexes to tables if you need to access under 5% of their data, except in the case of fairly small tables (which are more expediently searched in full whether or not you need much data). Do not include * in your SELECT statement queries unless necessary to fetch data, as this symbol will load the system. Use filters in WHERE clauses to restrict the size of the data set. Conversely, in a column-oriented system, only select the columns you need for the query. Cull unnecessary tables from query statements. Occasionally developers may forget to remove JOINS that do not serve the query. While this can be innocuous in the testing stage, once the system goes into effect, JOINS to tables that do not contribute to the retrieved data can greatly increase processing time. Utilize EXISTS in subqueries. This communicates to Oracle that it can stop the search, rather than complete a full-table scan by default, when it finds the match. Do not use indexes to tables that undergo more UPDATE or INSERT operations, as indexes can slow data input. In that same vein, you might consider dropping your indexes when striving for batch updates or insertions. In this case, it might be best to recreate the indexes after a single batch event, or simply avoid indexes on tables frequently experiencing batch data loads to begin with. Don't mix data types, and do not convert numbers to characters. Their comparison can slow operations and impact performance. In some scenarios, it may be easier to create a new field than to perform a calculation function on a JOIN or WHERE clause. In this situation, the new field would contain the calculated value, which the statement would SELECT rather than calculate itself. To accomplish this, the person tuning the code would have to have permission to alter the datasets, of course—but this should be no problem for a DBA or other IT administrator. More broadly, align your SQL statements and datasets. Basically, comb through SQL syntax to ensure that you've written your statements in ways that match up with the data structure and allow easy access. Institute protocol by using procedures instead of individual statements. A procedure is a collection of statements, and they downsize the cost of executing a recurring query. For example, if you use an app that requires you to pull data weekly, this query may account for a significant amount of database activity. You could use a procedure to certify that the query executes quickly, and according to your execution plan, because database engines carry out procedures without optimizing them. Use global temporary tables (GTT) whenever possible to simplify complication summarization queries. By breaking up the work of work-intensive subqueries, GTT have been shown to improve database performance significantly. Use hints. Oracle makes a list of hints available online to help application designers and DBAs. The purpose, as explained by the database itself, is to allow administrators and developers to "alter execution plans" and "force various approaches." This allows designers who are tuning their SQL statements to take the reins from the optimizer in specific scenarios where the humans know more about the data than the optimizer. In these situations, they can ensure their execution plan gets carried out rather than overwritten by the optimizer, which might choose a data access path that does not optimize speed and performance. Finally, make tuning a routine. While you don't want to have to allot redundant labor to tuning the same queries over and over, performance tuning in SQL does require habitual maintenance to prevent database performance deterioration over time as both datasets and RDBMS software evolve. With this in mind, commit to normalizing and defragmenting the database routinely. Of course, all of this Oracle performance monitoring and assessment work is nearly impossible to do manually on an ad hoc basis. Plenty of useful and affordable software exists that can automate database performance tracking and help DBAs identify the source of performance problems. This saves IT departments a good deal of time. Rather than noting a lag and having to root through their code and database set up to find the root cause, they can check their software to drill down into the queries with the highest performance impact. Here are the Oracle database tools I rate most highly: Database Performance Analyzer SolarWinds® Database Performance Analyzer® (DPA) is one of the most powerful solutions for those wondering how to check SQL query performance in Oracle. The dashboard was designed with the user experience in mind, and it provides comprehensible, informative visualizations breaking down SQL statement activity—data which can be dense and difficult to parse on other platforms. © 2023 SolarWinds Worldwide, LLC. All rights reserved. The dashboard displays a time analysis graph which plots the day against total query response time (the amount of time between a query request and a query response; basically, the cumulative processing time and resource availability wait time). The bar for each individual day is color-coded by query statement, which allows DBAs to easily drill down into metrics for each query simply by clicking. DPA collates data in real time, so its information is up-to-date from one second to the next, and it shows alerts on the dashboard (or delivers them via mobile notifications or email) when it detects critical issues. This helps DBAs hierarchize their chaotic workload, taking care of potential problems and query wait times that have the biggest impact on system performance before focusing their attention elsewhere. This function is particularly useful because it focuses performance-tuning work—which can require high investment for low returns when mismanaged—on the priority: wait time. Many tools and even IT departments make the mistake of prioritizing resource utilization during performance tuning, but this can obfuscate the source of a performance problem. It's guesswork, more than anything—something which is not true a database performance analytics which cross-check specific queries, wait time events, I/O and storage stats, and database activity against one another to determine causality and find performance solutions. Basically, SolarWinds DPA is unparalleled in the way it consolidates and analyzes data to make it truly actionable. Its graph of worst-performing SQL statements enables users to learn the specific wait types and events that are causing application wait times. If they glean that a wait is due to the CPU, for example, DBAs have a place to start performance tuning: the actual root cause. Fixing this issue will immediately yield a performance boost with tangible benefits for network users and applications. Further, DPA improves group communication and institutional memory. Not only does DPA preserve historical data, but it produces custom reports, alerts, and metrics for DBAs to present internally to colleagues or externally to other departments or even clients. Idera SQL Diagnostic Manager Another excellent tool for those learning how to check Oracle database performance is the Idera SQL Diagnostic Manager. It includes some features similar to the SolarWinds DPA tool, like predictive alerts and customizable monitoring settings for multiple servers. I found that its main strength are its prescriptive suggestions. That is, if you have a performance problem, the tool will flag the issue and recommend a solution like executable script. This is particularly helpful for IT personnel who are still getting their footing in SQL query performance tuning, although of course they'll need their own expertise to troubleshoot complex bottlenecks. In my opinion, the interface is a bit too cluttered, but it's definitely a good option for Windows users. The only other problem I'd note is that it's configurable reports and monitoring don't feel as robust as they could be. Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Monitoring for performance issues and doing regular maintenance of databases will keep them tuned and performing better. Better-performing Systems Planning the initial database design, monitoring the performance, and maintaining it are the proactive steps to achieving better-performing systems. This applies to any database environment. Along with proactive monitoring and tuning for performance, DBAs need to deal with performance issues that arise at times. Oracle has a view of the sessions and a way to see the current statements that are running against the database. Looking at the queries running and validating the statistics that are currently on the tables are the beginning steps. With the cost-based optimizer, current statistics are important for the queries to choose the right plan. In OEM alerts, waits are shown with alerts, and OEM provides a list of top queries running and also shows if any process is being blocked. In Oracle, blocking normally is checked after figuring out what is running and validating the statistics. Indexes Indexes exist to help speed up queries. Having proper columns indexed can reduce the logical I/Os for queries. There are costs associated with data changes when indexes are involved. Maintenance requirements should also be considered. The performance gains of adding an index should be more than the cost of maintaining it. Also, it is important to know that too many indexes can add to performance issues instead of resolving them. Hence, indexes should be used selectively and their usage should be monitored appropriately. Index Monitoring By enabling index monitoring, we can see which indexes are being used in Oracle. The owner of an index can alter the index to enable monitoring and leave it on for a set period. The v$object_usage table will show whether an index is used. SQLPLUS> alter index IDX1_EMP_DEPT monitoring usage; With this, the index is altered. SQLPLUS> select empno from emp where deptno=10; The rows are returned. SQLPLUS> select index_name,used, monitoring
from v$object_usage; INDEX_NAME USED MONITORING ————————- ———- ———- IDX1_EMP_DEPT YES YES The v$object_usage table has two other columns that show the time when the monitoring is started and when it is stopped. Meanwhile, to end the monitoring of an index, we have to use this statement: alter index index_name no monitoring usage Index monitoring will not track how many times an index is used, but it does offer a way to find out if there are unused indexes on a table. Index Types Indexes are definitely a useful tool for improving access to data in the database. Understanding which type of index is being used and how to improve that index will help in performance tuning. Knowing how various index types affect data changes and improve SELECT statements will help us decide if the benefits of an index outweigh the costs of retaining it in place. Primary Key Indexes: The primary key index is created for a table when a constraint is added, and we can either use an existing index or create a new one. Function-based Indexes: Oracle's function-based index type can dramatically reduce query time. They are useful for large tables even with simple functions like UPPER to do string comparisons. Indexes for Views: Views use indexes on their associated tables to build information, but there might be a need for an index for selecting from a view. Oracle has materialized views, similar to views, but are a snapshot of the data. These materialized views can have functions and aggregations, along with subqueries and other views, including self-joins. Bitmap Indexes: Bitmap indexes are stored differently than b-tree indexes. Instead of storing the row ID, a bitmap for each key is used. Because of this, these indexes are typically smaller in size and are useful for columns that have a low cardinality. Reverse Key Indexes: They are used as a nice little trick to spread out index blocks for a sequenced column. Reversing the numbers will allow the index to have different beginning values and use different blocks in the index b-tree structure. This is especially useful for RAC environments. When we are doing inserts, the reverse index will minimize the concurrency on the index blocks. Partitioned Indexes: Partitioning is a useful way to tune a large database environment. Oracle offers options for partitioning tables, such as LIST, HASH, RANGE, and COMPOSITE. The partition key is how the table is partitioned. We can create partitioned indexes for these tables. An index can be a locally partitioned index based on the partition key and setup used for each partition. Local indexes are easier to manage because they are handled with each partition, as partitions might be added, dropped, or merged. Invisible Indexes: Invisible indexes are hidden from the optimizer, but not from being maintained, so as rows are changed. One reason to use an invisible index is to test the performance of the queries without an index. Locking Holding locks on a database object will also cause another concurrent session to wait. Waiting to acquire a lock or perform a transaction could even cause blocking, depending on the locks required to perform a select transaction. A deadlock is when two or more users are waiting to access data locked by each other. When the deadlock occurs, Oracle chooses the victim, rolls back the transaction, and allows the other process to continue. Because of how Oracle handles locking, blocking is not always the first area that we should check for performance unless the application is trying to explicitly handle the locking outside Oracle. Access outside of the application, such as using query tools for ad-hoc queries, could open a transaction, and since the flow of the query is waiting on the user, Oracle Database would also wait on the user and hold onto the locks. So, if an UPDATE, INSERT, or DELETE statement is open in such a tool, there is no auto-commit that will release the locks. If the user does not issue a commit or rollback, this would leave an uncommitted transaction open, which could block others. Current Activity Views Oracle has various system views that provide the current session and wait for information. These are very helpful for performance tuning and troubleshooting. Current Sessions Obviously, when there are performance issues, it is necessary to take a look at the current sessions on the database. There is no sp_who, sp_who2, or sp_lock in Oracle, but there is the v$session view. This view shows which sessions are active. We can join this with another view to see all the queries that a session is running. Activity Monitors There are statistics gathered as part of Automatic Workload Repository (AWR) to provide reports for analyzing the health of the database and look for performance issues. Historical views are based on snapshots that have been gathered. Viewing these areas of activity can help us troubleshoot performance issues by pointing to an area that might be responding slowly or experiencing an overload, such as too many physical I/Os or hard parsing of SQL statements. The Instance Activity section shows values since the database has been up and running or back until the last snapshot that is available. Waits Another area to check into Oracle for performance issues is waiting for events. This information will be available in OEM views. The SQL*Net message from the client event is the wait for the client to tell the database server to do something. It is just waiting for instructions and really isn't contributing to issues. There might be applications that open sessions and then just wait for responses before getting the data from the database. Automatic Workload Repository Automatic Workload Repository (AWR) contains significant information that can be helpful when it comes to tuning the database environment. The database takes regular snapshots to get information about the database settings and the workload in the environment and stores it in AWR metadata tables (WRM$_) and historical statistics tables (WRH$_). In Oracle Database 11g, these reports and information are part of Oracle Diagnostic Pack, which provides an automated gathering of the information and ways to pull the information out of the workload and also history tables for review and evaluation of performance issues. We can also create baseline templates to compare the information. AWR Reports AWR reports have information about different waits. The reports list the top waits, providing a quick way to determine the areas where to start looking for bottlenecks. AWR reports can be viewed in OEM. The reports are based on snapshot times. If different intervals are needed, different reports can be generated. In OEM, we can view the details in the list or see the reports in the HTML format. Active Session History View The Active Session History (ASH) view has information about waits and events based on the sessions that are occurring in the database. The TIME_WAITED column shows the actual time waited for an event and will be updated when the event is completed. The WAIT_TIME column information matches up with the v$session_wait view. When the wait time is shown as zero, then the session is currently waiting; non-zero values indicate the session's last wait time. Library Cache for SQL Statements Oracle will first look in the library cache for code to be executed against the database, so there is no additional load into memory if the code is already there. The plans are also available there, so it is beneficial to be able to reuse SQL that is available in the library cache. AWR reports show a library cache hit ratio to indicate how much of the code is found in the cache and available for reuse. One reason for not finding the code in the library cache is that the cache is too small to hold all statements; if there are a lot of ad-hoc statements, it might be hard to hold all of them. Another reason for it could be due to the use of literal values instead of bind variables in the code. There is also a parameter that can help us make the code seem similar enough so that it can be reused: CURSOR_SHARING. This parameter can be set to one of the following: EXACT: This makes the code match exactly. Using this value will result in either a larger library cache/shared pool or a very low hit ratio of the library cache if literal values are used in the code and can't be matched up. FORCE: This will force a substitute of a literal into a bind variable to reuse the code. SIMILAR: This will allow Oracle to decide what to bind so that the code can be reused. Check out other related SQL blogs-