April 2006 Volume II Issue 2

ORAtips Journal

ORAtips On Database

Oracle®'s Approach to Performance Tuning Part II

By Darrick Addison

Editor's Note: Darrick Addison concludes his two part series on Oracle Performance Tuning with this article. In his first article he discussed the pros and cons of 10g tuning and presented well thought-out and well practiced answers brought about from experience to the golden question "What do I look for and how do I make it better?" He rounds out the topic with an examination of tuning methodology and issue resolution, exploring the common parameters every DBA responsible for performance management should consider and review. Darrick summarizes it best, writing "discipline needs to be exercised in setting reasonable tuning goals and ceasing all efforts when the tuning goals are achieved."

Haphazard attempts to increase the amount of memory consumed by the major components of the SGA can and will cause performance degradation.

Introduction

Oracle Performance management and tuning is part art and part science. For practical purposes, it can be further classified into two types, i.e., proactive and reactive performance management. Proactive performance management involves designing and developing the performance architecture for a system during the early stages of an implementation. This involves hardware selection, performance and capacity planning, mass storage system selection, I-O sub-system configuration and tuning (i.e., RAID), and tailoring the various components to suit the complex needs of the application and the Oracle database. The reactive component involves performance evaluation, troubleshooting, tuning, and fine-tuning an Oracle environment within the boundaries of the existing hardware and performance architecture.

The primary purpose of this paper is to provide a methodology for performance tuning Oracle and a set of core (minimal) parameters and components that require configuration and tuning.

Tuning Methodology

So, is there a method behind this madness? Arbitrary efforts are undertaken, where memory is just thrown at the Oracle System Global Area (SGA), in the hope that "the system performance problem will cure itself". Haphazard attempts to increase the amount of memory consumed by the major components of the SGA can and will cause performance degradation. In some production environments, it has been observed that excessive memory for the shared pool area has caused increased parse times, hanging of SQL statements, and in some cases a serious case of library cache latch contention.

Benchmark Oracle's Health (Before & After)

Performance snapshots of "before and after images" need to be captured during peak activity periods. These snapshots should not be taken immediately after instance startup, as it will skew the statistics. Also, snapshot(s) should be taken at 15-minute intervals during peak periods, as the relevant performance problems/issues may not "surface" if the intervals are longer.

A given snapshot is ended by running the script $ORACLE_HOME/rdbms/admin/utlestat.sql. On completion of execution of utlestat.sql, a report file called report.txt is created in the current directory with the system's performance statistics.

Troubleshooting the Problem

You begin the troubleshooting effort by looking at v$system_event to find out the events on the system. The next step is to look at v$session_event for the sessions that are causing and/or experiencing the wait events. The final step is to get the details behind the event from v$session_wait. Concurrent to this effort, a similar drill down should be done from the O-S by analyzing core CPU, Memory, and I-O related statistics. The two troubleshooting efforts should then move towards one another, and when they meet, the bottleneck is right there.

The First Step – V$System_Event

The v$system_event view provides a bird's eye view of all the events in an Oracle system. Even though it does not contain session-level specific information (current or the past), it does sum all the waits, since the last time the instance was bounced.

The statistics in this dynamic performance view is reset to zero on instance startup.

The Second Step – V$Session_Event

The v$session_event view provides the same information as v$system_event at the session level, plus relevant session information (i.e., SID, etc). This view allows the drill down from the "system-wide events" to the session level, to determine which session is causing and/or experiencing a given wait event.

The Final Step – V$Session_Wait

The v$session_wait view provides low-level drill down information at the session level for a given event. Unlike some of the other views, which display totals, this view displays session level wait information "real time".

Putting It All Together

As presented above, using session wait statistics helps one to pursue a different and more productive performance management effort. The key is the method of drilling-down to the core of the problem. So if you start with v$system_events view and determine that your top waits are for the event – db file sequential read (Wait Event for Index Scans…believe me, it is for Index Scans, even though it sounds like it is for Full Table Scans). On drilling-down to the v$session_events view, we are able to determine the sessions involved in the wait event.

The next step is to look at v$session_wait view for the relevant information, i.e., P1, P3, State, Wait_Time, & Seconds_In_Wait. This effort needs to continue until all of the contention related events are unearthed. At the same time, one has to determine the contributing SQL for this wait event. Remember, 80% of our problems are with SQL. In a parallel effort, the O-S statistics need to be analyzed, and that effort should be moving towards the Oracle environment. Nine out of ten times, if you have an index block or full-table scan related wait, the I-O statistics at the O-S will show their equivalent event – "wait of I-O". The device for this wait can then be ascertained and the O-S level drill down can begin.

Moral of the Story:

Don't fully depend on performance tuning a system with cache-hit ratios alone. If the database buffer cache hit ratio is low and you are beginning to get alarmed, stop and look at the session waits. If there are no wait events, then your perception of a performance problem is unfounded. On the flip side, if your cache-hit ratios are in the upper 90s and you find sessions waiting for events, don't just sit back thinking that everything is fine, because in reality it is not, and something is brewing.

Don't fully depend on performance tuning a system with cache-hit ratios alone.

DB_WRITERS

This parameter can be set for all Oracle implementations that are supported by file systems, and where Direct I-O is unavailable. This need not to be used with raw partitions, as asynchronous I-O is more optimal. It is recommended to set this parameter up to 2 * # of independent disk drives/volumes.

ASYNC_READ/ASYNC_WRITE/ DISK_ASYNCH_IO

Asynchronous I-O is the process of issuing an I-O call and not blocking until the operating system relays a confirmation about completing the I-O request (more relevant to writes rather than reads).

When asynchronous I-O is configured within Oracle, the operating system sends the final confirmation that a write is complete to the DBWR process, only if the data is physically on disk. But in the mean time, DBWR does not have to block and wait for the confirmation. When I-O sub-systems are configured with a "write cache", the operating system has no way of determining whether the data is physically down on disk, as low-level firmware in the I-O sub-system perform the write to disk from the cache.

COMPATIBLE

This parameter defines the "code tree" for the Oracle executables. This needs to be set to the current release of the Oracle software installed, unless instructed otherwise by Oracle World Wide Support, to get around some platform/version-specific bugs.

DB_FILE_MULTIBLOCK_READ_ COUNT

This parameter determines the I-O chunk size for reads while performing full-table-scans and full/range-index-scans. This parameter needs to be set in compliance with the largest I-O chunk size that the O-S can sup-

port. (If O-S chunk size is 128Kb and the database block size is 8Kb, then DB_FILE_MULTIBLOCK_READ_COUNT is usually set to either 16 or 32.)

SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE

The first parameter specifies the maximum amount of memory, in bytes, that is available for the sorting phase of a sort. The second parameter sets the maximum amount of memory, in bytes, that is available for the fetch phase of a sort. Both parameters are usually set to the same value. Sorts requiring more space than SORT_AREA_SIZE need to allocate and use temporary segments on disk. It is important to ensure that the ratio of (sorts on disk/sorts in memory) is less than 1%.

DB_BLOCK_LRU_LATCHES

This parameter defines the number of latches that are configured for the LRU list(s) of the database buffer cache. It can be set to its maximum value of (12 * # of CPUs) without any degradation in performance, and one can quit worrying about database buffer cache LRU latch contention.

_DB_BLOCK_HASH_BUCKETS

This parameter defines the number of hash buckets that are available for the database buffer cache. This has a direct impact on the length of the chain that a server process has to traverse to identify and read a given block in the database buffer cache. This parameter can be set to the value of DB_BLOCK_BUFFERS, so that every block will be in its own chain.

_DB_BLOCK_WRITE_BATCH

This parameter controls the chunk size for normal writes in the database. This parameter needs to be configured optimally such that the I-O sub-system is not overburdened, but the database writes that occur are completed within a reasonable amount of time.

DB_BLOCK_CHECKPOINT_BATCH

This parameter controls the chunk size for writes that occur during a checkpoint in the database. It needs to be configured optimally such that the I-O sub-system is not overburdened, but the database writes that occur are completed within a reasonable amount of time.

PARALLEL_MIN_SERVERS

This parameter defines the minimum number of parallel query slaves that are launched and retained from instance startup. This parameter can usually be set at (2 * # of CPUs).

PARALLEL_MAX_SERVERS

This parameter defines the maximum number of parallel query slaves that can be simultaneously launched during the life of an instance. This parameter can usually be set at (4 * # of CPUs).

PARALLEL_SERVER_IDLE_TIME

This parameter defines in minutes the amount of time that a parallel query slave can be idle before it is de-allocated. This parameter needs to be set based on the activity and usage of Parallel Query on the system, but can be initially set to 10-15 minutes.

PARALLEL_SERVER_MIN_PERCENT

This parameter defines the minimum number of parallel query slaves that need to be available for a given operation to be executed in parallel. If this parameter is set to 0, the default, and there are no available parallel query slaves for a parallel operation, then the operation is executed serially.

I-O tuning should be done after gaining a full understanding of the workings of Oracle and studying the architecture of the Oracle RDBMS.

I-O Tuning

I-O tuning is a critical component of system tuning and it involves among other tasks, spreading hot files across multiple drives/volumes, employing optimal striping methodologies, identifying I-O-sub-system bottlenecks, identifying controller bottlenecks, and choosing the optimal level of RAID, based on the type of application. I-O tuning should be done after gaining a full understanding of the workings of Oracle and studying the architecture of the Oracle RDBMS. I-O tuning should always be preceded and succeeded by I-O statistics monitoring, such as average service time, IOPS, average length of the disk queue, etc.

Conclusion

Managing and tuning Oracle Database Performance efforts requires adherence to a systematic methodology to ensure that all the core avenues and issues are addressed. Most of the issues can be proactively managed. Needless to say, the effort to configure systems with a balanced amount of hardware is of great importance. So let's face the fact – 80% of all system performance problems can be fixed by writing optimal SQL. This article has attempted to provide you with all the core facts needed to cover the remaining 20%. Also, discipline needs to be exercised in setting reasonable tuning goals and ceasing all efforts when the tuning goals are achieved. Knowing where one is going is important, but even more important is, knowing when one has reached the destination.

...80% of all system performance problems can be fixed by writing optimal SQL.

Darrick Addison, ASC Technologies, LLC - Darrick works as a Senior Software Engineer/Consultant for his consulting company, ASC Technologies, LLC. He has been designing and developing custom software applications since 1993. He has worked on the design and development of software ranging from database applications, network applications (TCP/IP client/server), GUI applications, and embedded systems applications in various commercial and government environments. He currently holds a BS degree in Computer Science and is pursuing his Master's degree in Computer Science/Telecommunications at Johns Hopkins University. Darrick may be contacted at Darrick.Adison@ERPtips.com.

This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips. For training, consulting, and articles on JD Edwards or SAP, please visit our websites: www.JDEtips.com and www.SAPtips.com.