Optimal Oracle9*i* on HP-UX 11i*

Sanhita Sarkar
Oracle Corporation,
500 Oracle Parkway, M/S 401ip3
Redwood Shores, CA 94065
Sanhita.Sarkar@oracle.com
Phone: (650) 506-4611
Fax: (650) 413-0166

Keywords and Phrases.- Oracle9*i*, HP-UX 11.0, HP-UX 11i, Performance tools, HP C compiler and linker, profile based optimization, asynchronous I/O, lightweight timer, benchmarks, tuning, SQL, etc.

ABSTRACT

It is the joint task of any database and system vendor to tune their respective features in order to achieve the desired level of performance. Oracle9*i* is a highly optimizable software product taking advantage of the platform-specific features and capabilities. This paper discusses various features and aspects of performance tuning of Oracle9*i* on HP-UX 11i platform. The most common types of performance bottlenecks are CPU, memory and disk I/O contention. Performance tuning involves detecting and solving these bottlenecks at both system and database level This paper first tabulates the different HP-UX and Oracle performance tools and then describes the ways to use some of these tools in detecting and tuning each of these bottlenecks. It also enlists some of the new performance features like optimizer hints and initialization parameters in Oracle9*i* that may help in enhancing the overall server performance. During the course of the discussion, it refers to several tunable Oracle features like the database buffers, redo buffers and shared pool and also recommends tuning at SQL level. As operating system and database level tuning are not always enough, one technique for improving performance is to create efficient machine code and optimal programs by optimizing at the compiler and linker levels. This paper mentions the importance of proper utilization of HP C compiler and linker flags for building an optimal Oracle9*i* executable on HP-UX 11i. As optimality of oracle executables on HP-UX 11.0 is guaranteed to be forward compatible, all flags/options used for an optimal oracle build on HP-UX 11.0 should work on HP-UX 11i as well. In addition to the above, designing and profiling the application for optimality also enhances performance in general. This paper mentions the advantage of Profile-based optimization of Oracle9*i* on HP-UX. It also recommends configuring the system kernel parameters and granting some privileges best suited for an optimal execution of Oracle9*i* on HP-UX. Also described in this paper are some of the HP platform-specific new enhancements in Oracle9*i*. These are the enhanced features of asynchronous I/O; implementation of the HP Lightweight timer; enabling the SCHED_NOAGE process scheduling policy, etc. To summarize, this paper discusses some methodologies towards a well-tuned HP-UX system with an optimal Oracle9*i* database and an efficient application running against it.

1. INTRODUCTION

Performance strategies vary in their effectiveness, the type of applications running against an existing database and also on system level tuning. Well-designed applications play an important role in achieving better performance. These involve proper table design, optimal SQL statements, robust index designing and choice of good software. System performance is usually designed and built into a system. Performance problems usually occur as a result of some system resource being exhausted or the database not being optimally tuned. When a system resource is exhausted, the system is unable to scale to higher levels of performance. Performance tuning is based on careful planning and

* ***HP World 2001: Chicago, Illinois, USA***

design of the database, to prevent system resources from becoming exhausted and causing downtime. By eliminating resource conflicts, systems can be made scalable to the levels required by the business.

Tuning optimizes system performance by overcoming bottlenecks. The most common types of performance bottlenecks at the system and database level are ***a) CPU resource contention, b) memory contention*** and ***c) disk I/O contention***. One may use the different HP-UX performance monitoring tools and Oracle Tuning tools and new features to analyze the system activity and database performance. The different types of tools and features followed by ways to tune these bottlenecks using some of these tools and features are described in the ensuing sections.

1.1. HP-UX and Oracle Performance Tools and Features

This section enlists the several Oracle and HP-UX tools commonly used for detecting and monitoring performance problems both at system and database level. It also tabulates some of the new performance features including that of the optimizer in Oracle9*i* which are very useful for database level tuning.

1.1.1. HP-UX tools

Tables 1 and 2 describe the different HP-UX utilities and analysis tools useful for monitoring and analyzing system activity and database performance.

1.1.2. Oracle9i Tuning Tools and New Performance Features

Throughout its operation, Oracle maintains a set of "*virtual*" tables that record current database activity. These tables are called *dynamic performance tables*. Dynamic performance tables are not true tables and may be accessed only by the user *SYS* or *SYSTEM*. However, database administrators can query and create views on the tables and grant access to those views to other users. These views are sometimes called fixed views because they cannot be altered or removed by the database administrator.

The Oracle9*i* new features of the optimizer and other enhancements comprise the overall effort to optimize server performance [3]. Tables 3, 4 and 5 provide information on dynamic performance views, SQL tuning tools and performance packs and new 9*i* features from Oracle that can help investigate and analyze problems with system resources, database and application performance.

TABLE 1: HP-UX Standard Utilities

Utilities	Use
gprof	Creates execution profile for programs.
monitor	Monitors program counter and calls to certain functions.
netfmt	Formats binary trace and log data collected from network tracing
netstat	Displays statistics for network interfaces and protocols.
nfsstat	Displays statistical information about the NFS and RPC interfaces to the kernel.
nettl	Captures network events or packets by logging and tracing.
prof	Creates execution profile of C programs.
profil	Dumps program counter information into a buffer. It controls execution time profiling.
sar	Samples cumulative activity counters at specified intervals from the OS.
top	Displays top processes on the system and periodically updates the information.
vmstat	Reports process, virtual memory, disk, paging and CPU activity on HP, depending on the command switches.
iostat	Reports terminal and disk activity depending on the command switches.
swapinfo	Reports about swap space usage. Shortage of swap space may cause system hang and slow response time.

tusc	Provides system call tracing functionality when attached to a process.

TABLE 2: HP-UX Performance Monitoring Tools

Analysis tools	Use
GlancePlus/UX	Online diagnostic tool displaying dynamic information about the system's I/O, CPU, memory usage and also resource usage by individual processes.
HP PAK (HP Programmer's Analysis Kit)	Consists of two tools - a) Puma, that collects performance statistics during a program run and provides graphical analysis and b) Thread Trace Visualizer (TTV) that displays trace files produced by the instrumented thread library in a graphical format.

TABLE 3: Oracle Dynamic Performance Views

Fixed Views	Information Type from the View	Use
V$LOCK	Current State View	Locks currently held/requested on the instance
V$LATCH_HOLDER	Current State View	Sessions/processes holding a latch
V$OPEN_CURSOR	Current State View	Cursors opened by sessions on the instance
V$SESSION	Current State View	Sessions currently connected to the instance
V$SESSION_WAIT	Current State View	Different resources sessions are currently waiting for
V$MYSTAT	Summary since instance startup	Resource usage summary for your own session
V$SESSION_EVENT	Summary since instance startup	Session-level summary of all the waits for current sessions
V$SESSTAT	Summary since instance startup	Session-level summary of resource usage since session startup
V$DB_OBJECT_CACHE	Summary since instance startup	Object level statistics in shared pool
V$FILESTAT	Summary since instance startup	File level summary of the I/O activity
V$LATCH	Summary since instance startup	Latch activity summary
V$LATCH_CHILDREN	Summary since instance startup	Aggregate summary for each type of latch
V$LIBRARYCACHE	Summary since instance startup	Namespace level summary for shared pool
V$ROLLSTAT	Summary since instance startup	Rollback activity summary
V$ROWCACHE	Summary since instance startup	Data dictionary activity summary
V$SQL	Summary since instance startup	Child cursor details for V$SQLAREA
V$SQLAREA	Summary since instance startup	Shared pool details for statements/anonymous block
V$SYSSTAT	Summary since instance startup	Summary of resource usage
V$SYSTEM_EVENT	Summary since instance startup	Instance wide summary of resources waited for
V$UNDOSTAT	Summary since instance startup	Undo space summary for a ten minute interval

V$WAITSTAT	Summary since instance startup	Break down of buffer waits by class
V$PARAMTER and V$SYSTEM_PARAMETER	Information View	Parameters values for one's session. Instance wide parameter values
V$PROCESS	Information View	Server processes (background and foreground)
V$SQL_PLAN	Information View	Execution plan for cursors that were recently executed
V$SQLTEXT	Information View	SQL text of statements in the shared pool

TABLE 4: Oracle Tools for Performance tuning

Oracle Tuning tools	Purpose
EXPLAIN PLAN statement	Displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. It also helps one to understand the optimizer decisions and explains the performance of a query. **Usage: EXPLAIN PLAN FOR <SQL Statement>**
SQL Trace Facility and TKPROF	Allows one to accurately assess the efficiency of the SQL statements an application runs. **Usage: ALTER SYSTEM SET SQL_TRACE = true;** One needs to run **TKPROF** to translate the trace file created in the previous step into a readable output file.
Autotrace in SQL*PLUS	Automatically generates a report on the execution path used by the SQL optimizer and the statement execution statistics. The report is generated after successful SQL DML (that is, SELECT, DELETE, UPDATE and INSERT) statements. **Usage: SET AUTOTRACE ON** The **AUTOTRACE** report includes both the optimizer execution path and the SQL statement execution statistics.
Statspack	The Statspack package builds off the traditional **UTLBSTAT/UTLESTAT** tuning scripts. Statspack automates the gathering of data, stores data and statistics, and generates performance reports. Statspack takes "snapshots" of data to work with, letting one to choose the snapshot levels and thresholds to be used.

TABLE 5: Oracle9*i* - New Performance Features

Oracle9*i* New Performance Features	Purpose
FIRST_ROWS_*n* Optimization	Setting the initialization parameter **OPTIMIZER_MODE** to **FIRST_ROWS_*n*,** the optimizer uses a cost-based approach, regardless of the presence of statistics, and optimizes with a goal of best response time to return first *n* number of rows (where *n* can equal 1, 10, 100, or 1000).
Literal Replacement with bind variables **CURSOR_SHARING parameter**	The cost based optimizer now peeks at the values of user-defined bind variables on the first invocation of a cursor. When bind variables are used in a statement, it is assumed that cursor sharing is intended and that different invocations are supposed to use the same execution plan. A new **CURSOR_SHARING** parameter can now be set to **SIMILAR** to force similar statements to share SQL by replacing literals with system-generated bind variables. Replacing literals with bind variables improves cursor sharing with reduced memory usage, faster parses, and reduced latch contention.
Identifying Unused Indexes	One can find indexes that are not being used by using the **ALTER INDEX MONITORING USAGE** functionality over a period of time that is representative of one's workload.
System Statistics	For each plan candidate, the optimizer computes estimates for I/O and CPU costs. It is important to know the system characteristics to pick the most efficient plan with optimal proportion between I/O and CPU cost.
Optimizer Hints	The following hints are new with 9*i*: **NL_AJ, NL_SJ, CURSOR_SHARING_EXACT, FACT, NO_FACT** and **FIRST_ROWS_ n**.

Outline Editing	While the optimizer usually chooses optimal plans for queries, there are times when users know things about the execution environment that are inconsistent with the heuristics that the optimizer follows. By editing outlines directly, one can tune the SQL query without having to alter the application. The **DBMS_OUTLN** package (synonym for **OUTLN_PKG**) and the new **DBMS_OUTLN_EDIT** package provide procedures used for managing stored outlines and their outline categories.
CPU Costing	The optimizer now calculates the cost of access paths and join orders based on the estimated computer resources, including I/O, CPU, and memory.
Tuning Oracle-Managed Files	Oracle internally uses standard file system interfaces to create and delete files as needed for tablespaces, tempfiles, online logs, and controlfiles.
FAST_START_MTTR_TARGET Parameter	One may now specify in seconds the expected **"mean time to recover" (MTTR),** which is the expected amount of time Oracle takes to perform recovery for an instance.
SQL Working Memory Management	With release 9*i*, it is now possible to simplify and improve the way the PGA is allocated in DSS systems. There is an automatic mode to dynamically adjust the size of the tunable portion of the PGA memory allocated by an instance. The size of that tunable portion is adjusted based on an overall PGA memory target explicitly set by the DBA.

1.2. Analyzing and Tuning Performance Bottlenecks

This section will discuss in details the common bottlenecks, ways of detecting and tuning them using some of the HP-UX and Oracle tools listed in the previous section 1.1. For more information regarding tuning at system and Oracle level, please refer to [1], [4], [5], [6], [8] and [9].

1.2.1. Tuning CPU Resources

If one suspects a problem with CPU usage, it is recommended to check two areas: **a)** ***CPU Utilization by the system*** and ***b) CPU Utilization by Oracle.***

1.2.1.1. Tuning CPU utilization by the system

Oracle statistics report CPU use by Oracle sessions only, whereas every process running on one's system affects the available CPU resources. Therefore, tuning non-Oracle factors can also improve Oracle performance. HP-UX monitoring tools may be used to determine what processes are running on the system as a whole. If the system is too heavily loaded, it is better to check the memory, I/O, and process management areas described as follows.

The ***sar*** utility **(usage: *sar -uM <time> <interval>*)** on a HP-UX system helps determining per-CPU utilization as well as the average CPU utilization of all the active processors on the entire system in specified *"interval"*s of the *"time"* in secs. CPU utilization is described in statistics showing user time, system time, idle time, and time waiting for I/O. A CPU problem exists if idle time and time waiting for I/O are both close to zero (less than 5%) at a normal or low workload. Tuning system CPU resources include tuning the ***a) memory management issues; b) I/O management issues*** and ***c) process management issues.***

Memory management issues

Paging and Swapping: Paging and swapping consume a lot of CPU resources. One may use tools such as ***sar*** **(usage: *sar -w <time> <interval>*) or *vmstat*** **(usage: *vmstat -S <interval> <count>*)** to get the reports for paging and swapping in *"interval"*s of *"time" or "count"* number of times during an *"interval"*. One may need to either increase the total memory on the system or decrease allocated memory.

Oversize Page Tables: If the processing space becomes too large, it can result in very large page tables. This may use up lot of CPU time in loading large page tables.

I/O management issues

***Thrashing*:** One should ensure that the workload fits into memory, so the machine is not thrashing (swapping and paging processes in and out of memory). The operating system allocates fixed portions of time during which CPU resources are available to one's process. If the process wastes a large portion of each time period checking to be sure that it can run and ensuring that all necessary components are in the machine, then the process might be using only 50% of the time allotted to actually perform work.

Client/Server Round Trips: The latency of sending a message can result in CPU overload. An application often generates messages that need to be sent

through the network over and over again, resulting in significant overhead before the message is actually sent. To alleviate this problem, it is recommended to batch the messages and perform the overhead only once. For example, one can use array inserts, array fetches, and so on.

Process management issues

***Scheduling and Switching*:** The operating system can spend excessive time scheduling and switching processes. So it is better to examine the way in which one is using the operating system, because one could be using too many processes.

***Context Switching*:** Due to operating system-specific characteristics, the system could be spending a lot of time in context switches. Context switching can be expensive, especially with a large SGA. Programmers often create single-purpose processes, exit the process, and create a new one. Doing this re-creates and destroys the process each time; thus necessitating page table rebuilds each time. Such logic uses excessive amounts of CPU, especially with applications that have large SGAs (SGA stands for shared global area) and hence large number of pages and page table entries. The problem is aggravated when one pins or locks shared memory, because one has to then access every page. For example, with 1 gigabyte SGA, one might have page table entries for every 4K, and a page table entry might be 8 bytes. So one could end up with (1G/4K) * 8B entries. This may become very expensive, because one needs to load so many entries each time the page table is rebuilt.

1.2.1.2. Tuning CPU Utilization by Oracle

This section explains how to examine CPU problems caused by Oracle processes running on the system. Three dynamic performance views provide information on Oracle processes: a) ***V$SYSSTAT*** shows Oracle CPU usage for all sessions. The statistic ***"CPU used by this session"*** shows the aggregate CPU used by all sessions. b) ***V$SESSTAT*** shows Oracle CPU usage per session. One can use this view to determine which particular session is using the most CPU. c) ***V$RSRC_CONSUMER_GROUP*** shows CPU utilization statistics on a per consumer group basis if one is running the Oracle Database Resource Manager. For example, if one has eight CPUs, then for any given minute in real time, one has eight minutes of CPU time available. At any given moment, one may want to know how much time Oracle has used on the system. So, if eight minutes are available and Oracle uses four minutes of that time, then one knows that 50% of all CPU time is used by Oracle. One then needs to identify the processes that are using CPU time. Possible areas to research why the processes are using so much CPU time and the resolution include, but are not limited to the following: ***Reparsing SQL Statements, Checking Read Consistency, Scalability Limitations within the Application, Detecting Wait Time and Latch Contention.***

Reparsing SQL Statements

When Oracle executes a SQL statement, it parses it to determine whether the syntax and its contents are correct. This process can consume significant overhead. Once parsed, Oracle does not parse the statement again unless the parsing information is aged from the memory cache and is no longer available. Ineffective memory sharing among SQL statements can result in reparsing. One may use the following procedure to determine whether reparsing is occurring: Get the ***"parse time cpu"*** and CPU figures used by this session from the ***"Statistics"*** section of the ***utlestat*** report or from ***V$SYSTAT***, as follows:

```
SELECT * FROM V$SYSSTAT
WHERE NAME IN('parse time cpu', 'parse
time elapsed', 'parse count (hard)');        [1]
```

From the report generated by SQL statement (1) one can detect the general response time on parsing. The more one's application is parsing, the more contention exists, and the more time the system spends waiting.

a) If ***"parse time cpu"*** represents a ***large percentage*** of the CPU time, then time is being spent parsing instead of executing statements. If this is the case, then it is likely that the application is using literal SQL and not sharing it, or the shared pool is poorly configured.

To find the frequently reparsed statements one can query ***V$SQLAREA*** as follows:

```
SELECT SQL_TEXT, PARSE_CALLS,
EXECUTIONS
FROM V$SQLAREA
ORDER BY PARSE_CALLS;                        [2]
```

and try tuning the statements with the higher number of parse calls.

b) If ***"parse time cpu"*** is only a ***small percentage*** of the total CPU used, then one should determine where the CPU resources are going. There are several things one can do to help with this.

- Find statements with large number of ***"buffer_gets"***, because these are typically heavy on CPU. The following statement finds SQL statements that frequently access database buffers. Such statements are probably looking at many rows of data. One may do as follows:

```
SELECT ADDRESS, HASH_VALUE,
BUFFER_GETS, EXECUTIONS, BUFFER
GETS/EXECUTIONS "GETS/EXEC",
SQL_TEXT
FROM V$SQLAREA
WHERE BUFFER_GETS > 50000
AND EXECUTIONS > 0
ORDER BY 3;                    [3]
```

The report from SQL statement (3) shows which SQL statements have the most ***"buffer_gets"*** and using the most CPU. The statements of interest are those with a large number of gets per execution, especially if execution is high. It is beneficial to have an understanding of the application components in order to know which statements are expected to be expensive. The 50,000 cut-off value is an arbitrary starting point and should be increased or decreased gradually until the top 10 to 20 statements are listed. This statement does not highlight CPU-intensive PL/SQL blocks.

- After candidate statements have been isolated, the full statement text can be obtained using the following query, substituting relevant values for ***ADDRESS*** and ***HASH_VALUE*** pairs. One can do as follows:

```
SELECT SQL_TEXT FROM V$SQLTEXT
WHERE ADDRESS='&ADDRESS_WANTED'
AND HASH_VALUE=&HASH_VALUE
ORDER BY piece;                [4]
```

The statement can then be explained using ***EXPLAIN PLAN*** or isolated for further testing to see how CPU-intensive it really is. If the statement uses bind variables and if one's data is highly skewed, then the statement might only be CPU-intensive for certain bind values.

- Find which sessions are responsible for most CPU usage. The following statement helps locate sessions that have used the most CPU:

```
SELECT v.SID, SUBSTR(s.NAME,1,30)
"Statistic", v.VALUE
FROM V$STATNAME s, V$SESSTAT v
WHERE s.NAME = 'CPU used by this session'
AND v.STATISTIC# = s.STATISTIC#
AND v.VALUE > 0
ORDER BY 3;                    [5]
```

After any CPU-intensive sessions have been identified, the ***V$SESSION*** view can be used to get more information. At this stage, it is generally best to revert to user session tracing ***(SQL_TRACE)*** to determine where the CPU is being used.

- Trace typical user sessions using the ***SQL_TRACE*** option to see how CPU is apportioned amongst the main application statements. After these statements have been identified, one has the following *three* options for tuning them: a) Rewrite the application so that statements do not continually reparse; b) Reduce parsing by using the initialization parameter ***SESSION_CACHED_CURSORS*** and c) If the parse count and execute count are small, and the SQL statements are very similar except for the ***WHERE*** clause, then one might find that hard-coded values are being used instead of bind variables. Using bind variables will reduce parsing in this case.

Checking Read Consistency

A system might spend excessive time rolling back changes to blocks in order to maintain a consistent view. Let us consider the following scenarios: a) If there are many small transactions and a long query running in the background on the same table where the inserts are taking place, then the query might need to roll back many changes; b) If the number of rollback segments is too small, then the system could also be spending a lot of time rolling back the transaction table. A solution is to make more rollback segments or to increase the commit rate. For example, if one batches ten transactions and commit them once, then one reduces the number of transactions by a factor of 10; c) If the system has to scan too many buffers in the foreground to find a free buffer, then it wastes CPU resources. To alleviate this problem, the DBWR process(es) should be tuned to write more frequently. One can also increase the size of the buffer cache to enable the database writer process(es) to keep up.

Scalability Limitations within the Application

In most of this CPU tuning discussion, we have been assuming that one can achieve linear scalability, but this is never actually the case. How flat or nonlinear the scalability is indicates how far away from optimal performance a system is. Problems in one's application might be adversely affecting scalability. Examples of this

include too many indexes, right-hand index problems, too much data in the blocks, improper partitioning of the data. These types of contention problems waste CPU cycles and prevent the application from attaining linear scalability.

Detecting Wait Time

Whenever an Oracle process waits for something, it records it as a wait using one of a set of predefined wait events. One may check ***V$EVENT_NAME*** for a list of all wait events. Some of these events can be considered as idle events i.e., the process is waiting for work. Other events indicate time spent waiting for a resource or action to complete. By comparing the relative time spent waiting on each wait event and the ***"CPU used by this session"*** (see SQL statement (5)), one can see where the Oracle instance is spending most of its time. To get an indication of where time is spent, one may find the following steps useful:

- Review either the ***V$SYSTAT*** view or the wait events section of the ***UTLBSTAT/UTLESTAT*** report.
- Ignore any idle wait events. Common idle wait events include: Client message, SQL*Net message from client, PMON timer, SMON timer, Parallel query dequeue, etc.
- Ignore any wait events that represent a very small percentage of the total time waited.
- Add the remaining wait event times, and calculate each one as a percentage of total time waited.
- Compare the total time waited with the ***"CPU used by this session"*** figure (see SQL statement (5)).
- Find the event with the largest wait event time. This may be the first item one needs to tune.

Detecting Latch Contention

Latch contention is a symptom of CPU problems; it is not usually a cause. To resolve it, one must locate the latch contention within the application, identify its cause, and determine which part of the application is poorly written. In some cases, the spin count may be set too high. It's also possible that one process is holding a latch that another process is attempting to secure. The process attempting to secure the latch might be endlessly spinning. After a while, this process might go to sleep and later resume processing and repeat its ineffectual spinning. To resolve this, one may do as follows:

- Check the Oracle latch statistics. The ***"latch free"*** event in ***V$SYSTEM_EVENT*** shows how long processes have been waiting for latches. If there is no latch contention, then this statistic does not appear. If there is a lot of contention, then it might be better for a process to go to sleep at once when it cannot obtain a latch rather than use CPU time by spinning.
- Look for the ratio of CPUs to processes. If there are large numbers of both, then many processes are able to run. But, if a single process is holding a latch on a system say, with 10 CPUs, then reschedule that process so it is not running. But 10 other processes might run ineffectively trying to secure the same latch. This situation wastes in parallel some CPU resources.
- Check ***V$LATCH_MISSES*** to figure out where in the Oracle code most of the contention is taking place.

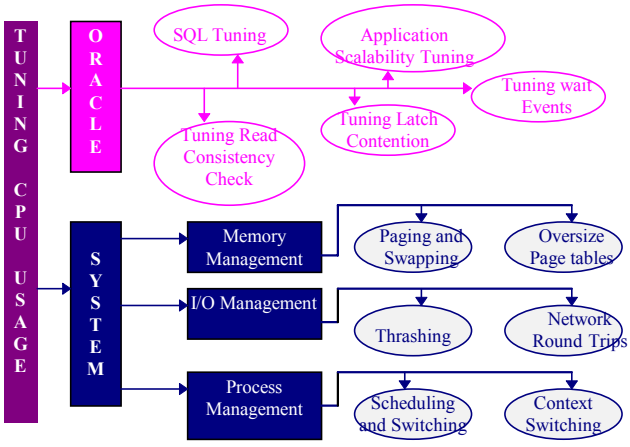

Figure 1. Tuning CPU resources both at system and Oracle level.

1.2.2. Tuning Memory Allocation

Oracle stores information in memory and on disk. Memory access is much faster than disk access; therefore, it is better for data requests be satisfied by accessing memory instead of accessing the disk. For best performance, it is better to store as much data as possible in memory. However, memory resources on an operating system are likely to be limited. Tuning memory allocation involves distributing available memory to Oracle memory structures. Oracle's memory requirements again depend on one's application. Therefore, it is desirable to tune memory allocation ***after*** tuning the application and SQL

statements. Also, it is recommended to tune memory allocation ***before*** tuning I/O. Allocating memory establishes the amount of I/O necessary for Oracle to operate. This section describes how to allocate memory to perform as little I/O as possible.

Resolving memory issues involve mainly ***a) Tuning Operating System Memory Requirements b) Tuning the Redo Log Buffer c) Tuning the Shared Pool*** and ***d) Tuning the Buffer Cache.***

1.2.2.1. Tuning Operating System memory Requirements

Tuning the operating system is centered around areas like ***a) Reduction of Paging and Swapping, b) Fitting the System Global Area into Main Memory*** and ***c) Allocating Adequate Memory to Individual Users.***

Reducing Paging and Swapping

An operating system can store information in real memory, virtual memory, expanded storage, or on disk. The operating system can also move information from one storage location to another. This process is known as paging or swapping. Many operating systems page and swap to accommodate large amounts of information that do not fit into real memory. Excessive paging or swapping can reduce the performance of many operating systems and indicates new information often being moved to memory. In that case, the system's total memory might not be large enough to hold everything for which one has allocated memory. So as a solution to this problem, it is recommended to either increase the total memory on the system or decrease the amount of memory allocated.

Fitting the System Global Area into Main Memory

Because the purpose of the System Global Area (SGA) is to store data in memory for fast access, the SGA should always be within main memory. If pages of the SGA are swapped to disk, then its data is no longer quickly accessible. On most operating systems, the disadvantage of excessive paging significantly outweighs the advantage of a large SGA. Although it is best to keep the entire SGA in memory, the contents of the SGA are split logically between hot and cold parts. The hot parts are always in memory, because they are always being referenced. Some cold parts can be paged out, and a performance penalty might result from bringing them back in. A performance problem likely occurs, however, when the hot part of the SGA cannot remain in memory. One may actually cause Oracle to read the entire SGA into memory when starting up the Oracle instance. Operating system page table entries are then pre-built for each page of the SGA. This setting can increase the amount of time necessary for instance startup because every process that starts must attach itself to the SGA. But it is likely to decrease the amount of time necessary for Oracle to reach its full performance capacity after startup. The advantage that this strategy of locking the whole SGA into memory can afford depends on the OS page size. HP-UX permits Oracle to set the largest virtual memory page size available (up to 1 Gigabyte) resulting in only a few pages to be touched to refresh the SGA.

Allocating Adequate Memory to Individual Users

On some operating systems like HP-UX, one may have control over the amount of physical memory allocated to each user. One must make sure that all users are allocated enough memory to accommodate the resources they need to use their application with Oracle. These resources include: The Oracle executable image, the SGA, Oracle application tools and Application-specific data. On HP-UX, Oracle software can be installed so that a single executable image can be shared by many users, thus reducing the amount of memory required by each user.

1.2.2.2 Tuning Oracle Memory Resources

Tuning Oracle memory resources mainly involves ***a) Tuning the Redo Log Buffer; b) Tuning the Shared Pool*** and ***c) Tuning the Buffer Cache.***

Tuning the Redo Log Buffer

When the Oracle Log Writer process (LGWR) writes redo entries from the redo log buffer to a redo log file or disk, user processes can copy new entries over the entries in memory that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy. The statistic ***REDO BUFFER ALLOCATION RETRIES*** reflects the number of times a user process waits for space in the redo log buffer. One may query as follows:

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME = 'REDO BUFFER
ALLOCATION RETRIES';          [6]
```

The value of ***REDO BUFFER ALLOCATION RETRIES*** should be close to zero. If this value increments consistently, then processes have had to wait for space in the buffer. The wait can be caused by the log buffer being too small or by checkpointing. If that's the case, one needs to increase the size of the redo log buffer by changing the value of the initialization parameter ***LOG_BUFFER*** or alternatively, improve the checkpointing or archiving process.

Tuning the Shared Pool

The shared pool contains the library cache of shared SQL requests, the dictionary cache, stored procedures, and other cache structures that are specific to a particular instance configuration. Proper sizing of the shared pool can reduce resource consumption. For example, parse time is avoided if the SQL statement is already in the shared pool; application memory overhead is reduced, because all applications use the same pool of shared SQL statements and dictionary resources; I/O resources are saved, because dictionary elements that are in the shared pool do not require disk access, etc. One might need to increase the shared pool size if the frequently used set of data does not fit within it. A cache miss on the data dictionary cache or library cache is more expensive than a miss on the buffer cache. So it is recommended to allocate sufficient memory to the shared pool before allocating to the buffer cache. One can determine the library cache and row cache (data dictionary cache) hit ratios from the following SQL queries (7). The results show the miss rates for the library cache and row cache. If the ratios are close to 1, then one does not need to increase the pool size.

```
SELECT (SUM(PINS - RELOADS)) / SUM(PINS) "LIB CACHE"
FROM V$LIBRARYCACHE;

SELECT (SUM(GETS - GETMISSES - USAGE - FIXED)) / SUM(GETS) "ROW CACHE"
FROM V$ROWCACHE;                                              [7]
```

Tuning the Buffer Cache

Physical I/O takes a significant amount of time, typically more than 15 milliseconds. Physical I/O also increases the CPU resources required, because of the path length in device drivers and operating system event schedulers. One's goal should be to reduce this overhead as much as possible by making it more likely that the required block is in memory. The extent to which one can achieve this is measured using the cache hit ratio. Within Oracle, this term applies specifically to the database buffer cache. One may monitor these statistics as follows:

```
SELECT NAME, VALUE
FROM V$SYSSTAT
WHERE NAME IN ('DB BLOCK GETS', 'CONSISTENT GETS', 'PHYSICAL READS');
                                                              [8]
```

The hit ratio for the buffer cache may be calculated using the following formula:

Hit Ratio = 1 - ((physical reads - physical reads direct) / session logical reads)

If the calculated hit ratio is low, say less than 60% or 70%, then one might want to increase the number of buffers in the cache to improve performance. To make the buffer cache larger, one needs to increase the value of the initialization parameter ***DB_BLOCK_BUFFERS***.

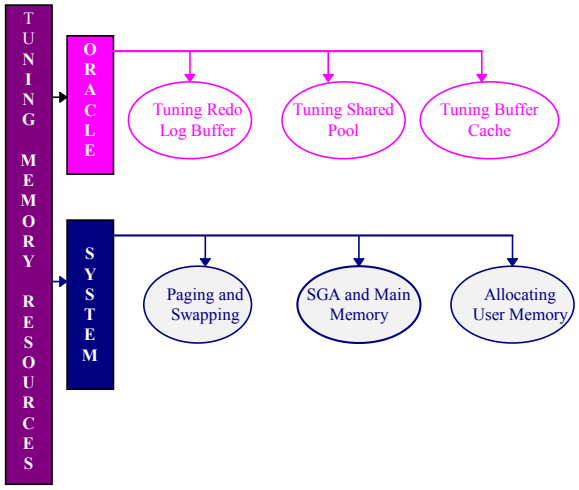

Figure 2: Tuning memory allocation both at system and Oracle level.

1.2.3. Tuning I/O Problems

Disk I/O contention is the result of poor memory management (with subsequent paging and swapping), or poor distribution of tablespaces and files across disks. The I/O load should be spread evenly across all disks. HP-UX monitoring tools may be used to determine what processes are running on the system as a whole and to monitor disk access to all files. Disks holding datafiles

and redo log files can also hold files that are not related to Oracle. In that case, one should try to reduce any heavy access to disks that contain database files. Tools, such as ***sar* (usage: *sar -d <time> <interval>*)** on HP-UX, lets one monitor the disk I/O activity for the entire system in the specified "*interval"s* of "*time"* in secs. Also the ***iostat*** utility **(usage: *iostat -t <interval> <count>*)** on HP-UX reports terminal and disk activity *"count"* number of times at specified *"interval"*s. It reports which disks are busy and helps balancing the I/O loads.

To check on Oracle I/O utilization, ***V$SYSTEM_EVENT*** can be queried by event to show the total number of I/Os and the average duration by type of I/O (read/write). With this, one can determine which types of I/O are too slow and may tune Oracle-related I/O problems. If the Oracle server is not consuming the available I/O resources, then the process that is using up the I/O should be determined for tuning. The view ***V$SYSTEM_EVENT*** may be further reviewed for the following events: ***db file sequential read, db file scattered read, db file single write,*** and ***db file parallel write***. These are all events corresponding to I/Os performed against the data file headers, control files, or data files. If any of these wait events correspond to high average time, then it is recommended to investigate the I/O contention using **sar** or **iostat** in order to look for busy waits on the device. The file statistics may help to determine which file is associated with the high I/O. Figure 3 shows the steps to tune I/O problems.

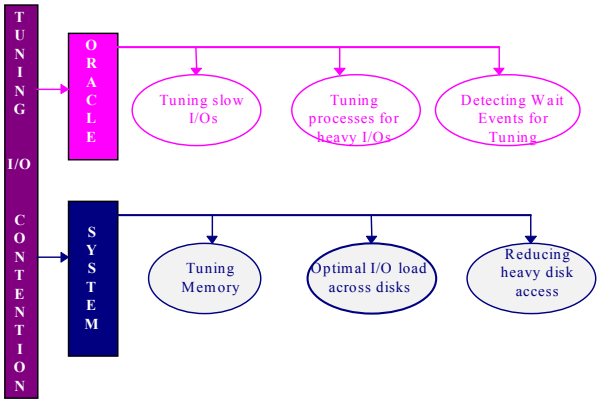

Figure 3: Tuning I/O contention both at system and Oracle database level.

2. CONFIGURING ORACLE9*i* ON HP-UX

This section discusses about different aspects that one needs to be aware of while trying to configure oracle9*i* on HP-UX. These are ***a) the set of required system privileges; b) recommended HP-UX kernel parameter settings*** and ***c) disabling data prefetch on Superdome systems.*** As the recommendations in a) and b) are applicable to both HP-UX 11.0 and 11i, I have used the term HP-UX to cover both. The discussion on data prefetch setting is applicable only to Superdome systems with HP-UX 11i.

2.1. Granting privileges to the group owning an Oracle9i executable on HP-UX

For Oracle9*i* version 9.0.1, the system administrators need to grant a few privileges to the group owning the Oracle executable. These privileges include ***MLOCK, RTSCHED*** and ***RTPRIO***.

The ***MLOCK*** privilege is required for Oracle 9*i* version 9.0.1 to execute asynchronous I/O operations using the HP asynchronous device driver. Without this privilege, users will see errors in the trace file as follows: ***"Ioctl ASYNC_CONFIG error, errno=1".***

A new scheduling policy from HP called the ***SCHED_NOAGE*** helps to enhance Oracle performance by scheduling Oracle processes in a manner so that they don't increase or decrease in priority or get preempted. To enable this policy, the user group owning the Oracle9*i* executable needs to have the ***RTSCHED*** and ***RTPRIO*** privileges. These privileges grant Oracle the ability to change its process scheduling policy to ***SCHED_NOAGE*** and also tell Oracle what priority level it should use when setting the policy.

In most of the cases, the group that owns the oracle executable is DBA. So the system administrator must execute as root to grant the above privileges to group DBA as follows:

 # setprivgrp dba MLOCK RTSCHED RTPRIO

To make the privilege persistent over reboots, the system administrator must enter the following into the file ***/etc/privgroup:***

dba MLOCK RTSCHED RTPRIO

If the file */etc/privgroup* doesn't exist, it should be created.

2.2. HP-UX Kernel Parameter Settings for Optimal Performance of 64-bit Oracle9i

The following Table 6 lists the different HP-UX kernel parameter settings recommended for a 64-bit Oracle9*i*. Certain kernel operating system parameters can be configured to fit specific system needs, resulting in better performance or more effective allocation of resources. On HP-UX 11i, some of the parameters are dynamically re-configurable. That way, users may alter the parameter settings related to process memory, shared memory, etc dynamically without going through a system reboot. For more details, please refer to [9]. The ideal value for each parameter is often determined by the system's particular hardware configuration, the specific mix of applications the system runs, and the trustworthiness of system users; factors that vary widely from system to system.

This paper attempts to provide reasonable parameter settings for running Oracle9*i* on HP-UX - the ones that we generally use for different OLTP and DSS benchmarks. One may find it necessary or beneficial to modify these settings to better suit the needs of the users of a particular system.

TABLE 6: HP-UX Kernel Parameter Settings recommended for running a 64-bit Oracle9*i*

Kernel Parameter	Recommended Setting	Purpose
nproc	4096	Maximum number of processes
ksi_alloc_max	(nproc * 8)	System-wide limit of queued signal that can be allocated
maxdsiz	1073741824	Maximum data segment size (bytes) A low setting may cause processes to run out of memory pretty fast
maxdsiz_64bit	2147483648	Maximum data segment size (bytes). A low setting may cause processes to run out of memory pretty fast
maxssiz	134217728	Maximum stack segment size (bytes)
maxssiz_64bit	1073741824	Maximum stack segment size (bytes)
maxswapchunks	(Available Physical Memory (in MB) / 2)	Maximum number of swap chunks where swchunk is the swap chunk size (1K blocks)
maxuprc	(nproc +2)	Maximum number of user processes
msgmap	(nproc +2)	Maximum number of message map entries
msgmni	nproc	Number of message queue identifiers
msgseg	(nproc * 4)	Number of segments available for messages
msgtql	nproc	Number of message headers
ncallout	(nproc + 16)	Maximum number of pending timeouts
ncsize	((8 * nproc + 2048) + VX_NCSIZE) where VX_NCSIZE is by default 1024	Directory Name Lookup Cache (DNLC) space needed for inodes
nfile	(15 * nproc + 2048)	Maximum number of open files
nflocks	nproc	Maximum number of file locks
ninode	(8 * nproc + 2048)	Maximum number of open inodes
nkthread	(((nproc * 7) / 4) + 16)	Maximum number of kernel threads supported by the system
semmap	((nproc * 2) + 2)	Max number of semaphore map entries
semmni	(nproc * 2)	Maximum number of semaphore sets in the entire system.
semmns	((nproc * 2) *2)	Maximum semaphores on the system.
semmnu	(nproc - 4)	Number of semaphore undo structures
semvmx	32768	Maximum value of a semaphore.
shmmax	~ available physical memory	Maximum allowable size of one shared memory segment. Should be big enough to hold the entire SGA in one shared memory segment. A low setting causes creation of multiple shared memory segments which may in turn cause performance degradation.
shmmni	512	Maximum number of shared memory segments in the entire system.

shmseg	32	Maximum number of shared memory segments one process can attach.
vps_ceiling	64	Maximum System-Selected Page Size (in Kbytes)

2.3. Disabling Data Prefetch on the HP Superdome for Oracle

HP Superdome systems have a data prefetch feature that might impair Oracle performance in update or insert intensive applications. Oracle Corporation and Hewlett-Packard Corporation recommend one to disable this feature for these types of applications.

Prefetching data usually improves application performance. However, the Oracle server is developed to run well on all HP systems, including those that do not include the prefetch feature. In update and insert intensive applications, enabling the data prefetch feature unintentionally creates contention on redo allocation and redo copy latches. This contention is increased as the number of processors is increased. Disabling the data prefetch feature helps to reduce redo latch contention.

HP Superdome systems can run multiple instances of the HP-UX 11i operating system on a single server by defining multiple partitions within a Superdome server. Each partition on a HP Superdome system acts as a logical server running a single instance of HP-UX. Each system can boot, reboot and operate independently of other partitions and hardware within the Superdome system. Each partition also has its own console.

To disable data prefetch on a HP Superdome system, one must disable the prefetch option in each partition on which the Oracle server is running. So to disable the prefetch option for each partition, one has to login to each partition and start a reboot in order to enter the ***BCH (Boot Console Handler)*** prompt. From the BCH prompt, one needs to enter the ***configuration menu (CO)*** and disable data prefetch ***(data prefetch disable)***. Once this is done, the partition has to go through a full reboot in order to reset this option for this partition. For more details, please refer to [2].

3. OPTIMIZING THE BUILDS OF ORACLE PRODUCTS ON HP-UX

The HP C optimizer can transform programs so that machine resources are used more efficiently thus dramatically improving application run-time speed. HP C performs only minimal optimizations unless specified otherwise. There are four major levels of optimizations: 1, 2, 3 and 4. Level 4 optimization can produce the fastest executable code and is a superset of other levels. But one may have to pay a penalty by choosing a high level optimization because of the increased compile-time memory and CPU usage with this choice. So one needs to use the desired level of optimization being aware of the trade-offs between compile-time penalties and code performance. One may activate optimizations of one's own choice using HP C command line options. One such desired type of optimization on HP-UX is the ***profile based optimization (PBO)*** which is a set of performance improving code transformations based on the run-time characteristics of the application. In addition to the above, one should use proper HP C optimizer flags during compile and link time as well as appropriate system libraries during link time in order to ensure optimal performance of oracle products on HP-UX.

3.1. Profile Based Optimization (PBO)

There are three steps involved in performing this type of optimization: ***a) Instrumentation, b) Flow Data Collection*** and ***c) Optimization***. Figure 4 shows the different stages towards a Profile Based Optimized Oracle product on HP-UX.

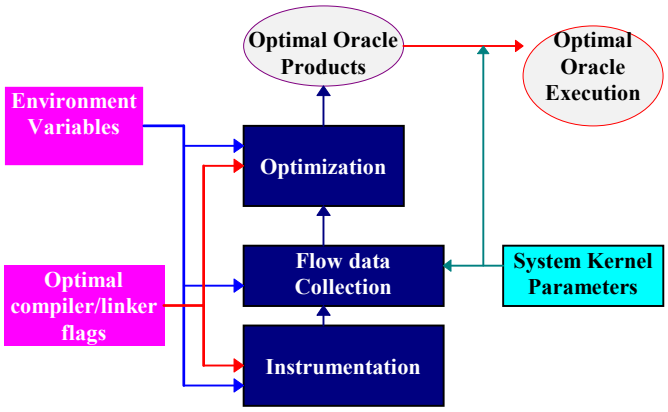

Figure 4: Phases for profile based optimization of an Oracle executable on HP-UX

The *instrumentation phase* inserts data collection code into the object program. The *data collection phase* creates and logs the profile statistics to a file, by default

called the ***flow.data***. It is a structured file that may be used to store the statistics from multiple test runs of different programs that one may have instrumented. The last phase called the *optimization phase* optimizes the program based on the collected run-time profile statistics. ***Profile Based optimization (PBO)*** has a greater impact on application performance at each higher level of optimization. *PBO* should be enabled during the final stages of application development. To obtain the best performance, it is recommended to re-profile and re-optimize the application after making any source code changes.

With Oracle9*i* version 9.0.1, lab tests indicate that proper *PBO*-ing provides up to 13% improvement in performance.

3.2. HP C Compiler/Linker flags and System libraries for 64-bit optimized Oracle products

Choosing the right HP C compiler and linker flags and HP/Oracle options is a procedure followed for building 64-bit optimized oracle products on HP-UX 11.0/11i. The HP-UX system libraries also play an important role while linking a 64-bit optimized oracle on HP-UX 11.0/11i. For details regarding the use of the different flags/options and system libraries, please refer to the HP-UX reference manuals in [9].

4. ENHANCEMENTS IN ORACLE9*i* FOR HP-UX

This section describes the different performance enhancements in Oracle9*i* release version 9.0.1 on HP-UX. These enhancements include implementation of ***asynchronous I/O to heap, asynchronous flag in Oracle shared global area, the new lightweight timer from HP*** and the ***SCHED_NOAGE process scheduling policy.***

4.1. Asynchronous I/O to heap for 64-bit Oracle 9.0.1 on HP-UX 11.0 and 11i

In Oracle9*i* release version 9.0.1 as well as in Oracle8*i* release version 8.1.7.0 and the intermediate releases on HP-UX, the Oracle server executes I/O operations from both shared memory and process-private region or heap using the HP asynchronous driver. This implementation enables Oracle parallel query slaves and database writers to execute I/O operations from shared memory as well as from local memory. As a result a fewer number of the above processes are sufficient to bring forth the desired performance with a lower memory usage especially for applications with large full table scans and index fast full scans. Lab tests show that the implementation of asynchronous I/O to heap provides up to 15% gain in DSS performance with the number of parallel query slaves reduced to half.

Before Oracle8*i* release version 8.1.7.0 on HP-UX, the Oracle server could execute I/O operations only from the shared memory using the HP asynchronous driver. As a result, the number of Oracle parallel processes required to achieve a desirable performance was a few times higher than what was recommended for a particular configuration. This also led to a higher percentage of memory usage on HP-UX compared to other platforms.

4.2. Asynchronous Flag in Shared Global area for 64-bit Oracle 9.0.1 on HP-UX 11.0 and 11i

Oracle9*i* release 9.0.1 on HP uses a non-blocking polling facility provided by the HP asynchronous driver to check the status of I/O operations. This polling is performed by checking a flag that is updated by the asynchronous driver based on the status of the I/O operations submitted. HP requires that this flag be in shared memory.

Oracle9*i* configures an asynchronous flag in the SGA for each oracle process. Oracle9*i* on HP has a true asynchronous I/O mechanism where I/O requests can be issued even though some previously issued I/O operations are not complete. This helps to enhance performance and ensures good scalability of parallel I/O processes. Lab tests show that this implementation provides up to 13% improvement in DSS performance.

Before Oracle8*i* release 8.1.7, the Oracle server was only able to execute I/O operations from shared memory using the HP asynchronous driver. Oracle8*i* release 8.1.7 executes I/O operations from both shared memory and process-private regions using the new HP asynchronous driver. However, I/O operations through the asynchronous driver are not asynchronous in nature. This is because Oracle8*i* must perform a blocking wait to check the status of I/O operations submitted to the asynchronous driver. Doing this causes some Oracle

processes, for example the database writer process, to essentially execute synchronous I/O.

4.3. HP Lightweight Timer for 64-bit Oracle 9.0.1

Prior to Oracle9i release version 9.0.1 on HP-UX, Oracle called the heavyweight HP-UX ***gettimeofday()*** system call to get the wall clock time and calculate elapsed time. This had a significant impact on Oracle performance, especially when the Oracle ***timed_statistics*** initialization parameter was set to true in order to collect timing information for tuning purposes. HP has recently designed a new high performance, lightweight timer library call function ***gethrtime()*** for both HP-UX 11.0 and 11i. The 64-bit version of Oracle 9*i* version 9.0.1 on HP-UX 11.0 and 11i uses this new library call to calculate elapsed time, greatly reducing the negative impact on RDBMS performance when the ***timed_statistics*** is set to true (lab tests show that the new library call provides up to 10% performance improvement over the previous implementation).

4.3.1. Requirement of Operating System Patches

Before running Oracle9*i*, one must make sure that the required operating system patches are installed on one's system. Running Oracle9*i* on unpatched versions of the HP-UX kernel will result in undefined and/or unresolved references to the ***gethrtime()*** library call.

For detailed information on the required patches, one should refer to the Oracle9*i* Quick Installation Procedure [7]. For patch availability and downloads, one should refer to the HP support web site.

4.4. Process Scheduling Policy SCHED_NOAGE for Oracle on HP-UX

By default, most processes run under the ***SCHED_TIMESHARE*** scheduling policy on HPUX. Each process has a priority and is given access to a CPU based on that priority. The scheduler keeps track of each process' priority. The priority of a process running on a CPU gradually degrades, while the priority of a process to be run increases. That way, no one process can monopolize the CPU. In a time sharing environment, this is the desired behavior most of the time. But in some cases, the standard schedule method may cause sub-optimal performance. If a running process has a lock on a resource and is preempted, a process that needs that resource may start running; realize that it can't acquire the resource and go right back to sleep again. In that case, it would have been better for the process with the lock on the resource to finish its work and relinquish the lock, instead of being preempted. This situation may often arise with Oracle processes on a multiprocessor system.

HP has a modified scheduling policy, referred to as ***SCHED_NOAGE***, that specifically addresses this issue. Unlike the normal time sharing policy, a process scheduled using ***SCHED_NOAGE*** does not increase or decrease in priority, nor is it preempted. This feature is suited to online transaction processing (OLTP) environments because OLTP environments can cause competition for critical resources. In laboratory tests, Oracle9*i* performance increased by up to 10 percent in OLTP environments using the ***SCHED_NOAGE*** policy. The ***SCHED_NOAGE*** policy creates little or no gains in decision support (DSS) environments because there is little resource competition in these environments. Because each application and server environment is different, it is recommended to test and verify whether one's environment benefits from the ***SCHED_NOAGE*** policy.

To allow Oracle9*i* to use the ***SCHED_NOAGE*** scheduling policy, the group that the Oracle software owner belongs to (DBA), must have the ***RTSCHED*** and ***RTPRIO*** privileges to change the scheduling policy and set the priority level for Oracle processes. To give the dba group these privileges: As the root user, one should enter the following commands:

- # ***setprivgrp dba RTSCHED RTPRIO***
- To retain these privileges after rebooting, one should create the /etc/privgroup file, if it does not exist on the system, and add the following line to it:

  dba RTSCHED RTPRIO
- One should add the ***HPUX_SCHED_NOAGE*** parameter to the initialization file for each instance, setting the parameter to an integer value to specify process priority levels. On HP-UX 11.0, the range is 154 to 255.

For more information on priority policies and priority ranges, one should see the rtsched (1) and rtsched (2) man pages and the HP documentation site [9].

5. PERFORMANCE ANALYSIS

This section analyzes the different performance enhancements in Oracle9*i* release version 9.0.1 on HP-UX 11.0/11i. Testing has been done internally at Oracle and HP performance labs and benchmarks have been carried out in order to measure the above enhancements. The discussion in the previous paragraphs correlates to the results from such measurements.

TPC-H and TPC-C baseline runs were taken on HP-UX 11.0 as well as 11i after installing the proper operating system patches corresponding to the enhancements. When **timed_statistics** was set to **true,** we have achieved up to 10% performance improvement using the lightweight timer**.** Use of the ***tusc*** utility during these runs, showed around 20 times reduction of calls to ***gettimeofday()*** with the new implementation. With the asynchronous flag implemented in Oracle9*i* release version 9.0.1, we have seen a 13% gain in the TPC-H benchmark performance. The mechanism of non-blocking polling for I/O completions has shown desirable scalability of Oracle parallel query slaves with a higher throughput from a terabyte size database. The ***SCHED_NOAGE*** process scheduling policy in the above mentioned version of Oracle9*i* has shown up to 10% improvement in performance of OLTP applications in a competitive scenario for critical resources, e.g., latches. Proper utilization of these enhancements and the required operating system patches have been discussed in details in [2] and [7].

6. CONCLUSIONS

We at Oracle continue to strive to bring our customers the most functional and best performing database available. In those efforts, we work very closely with HP in order to ensure the highest quality of products combined with very high performance systems. One can be assured that we will be working jointly for many years to deliver the products one needs in day to day production environments.

As an immediate future direction we have in mind several enhancements for the next Oracle9*i* release version 9.0.2 on HP-UX. For example, we will be working on further extensions of the lightweight timer for Oracle; use the enhanced features of the new HP linker for optimal linkordering and preloading shared libraries and also test out new patch bundles from HP. We will also be continuing experiments with Oracle *9iRAC (Real Application Cluster)* on HP-UX and work on other miscellaneous topics like vectored asynchronous I/O, performance of I/O executions on HP-UX using filesystems, e.g., JFS 3.*x* and so on.

7. REFERENCES

1. *Aronoff, E., Loney, K and Sonawalla, N. Advanced Oracle Tuning and Administration: McGraw Hill Publishers, Berkeley, California.*
2. *Oracle9i Administrator's Reference Release 9.0.1 for UNIX Systems: Oracle Corporation.*
3. *Oracle9i New Features, Release 9.0.1: Oracle Corporation.*
4. *Oracle9i Performance Guide and Reference Release 9.0.1: Oracle Corporation.*
5. *Oracle9i Performance Methods, Release 9.0.1: Oracle Corporation.*
6. *Oracle9i SQL Reference, Release 9.0.1: Oracle Corporation.*
7. *Oracle9i Quick Installation Procedure Release 9.0.1 for HP 9000 Series HP-UX: Oracle Corporation.*
8. *Sauers, R and Weygant, P. HP-UX Tuning and Performance: Concepts, Tools and Methods: Prentice-Hall Publishers, New Jersey.*
9. *Web Site for Technical Documentation from HP: www.docs.hp.com.*