A Fast Track to Oracle9 *i* Release 2 on HP-UX 11 *i*[*]

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 11*i*, Oracle9*i* Data Guard, Oracle9*i* Streams, Oracle9*i* Real Application Clusters, Cache Fusion, HP C compiler and linker, Profile Based Optimization, HP Hyper Messaging Protocol, HP HyperFabric, HP Caliper, Intel Itanium Processor Family, Asynchronous I/O, Lightweight timer, Process scheduling, Performance benchmarks, etc.

ABSTRACT

Oracle9*i* Database Release 2 greatly extends the technology of Oracle9*i* with key enhancements in certain areas like Data Guard, Oracle9*i* Streams, automated memory and space management, partitioning techniques, OLAP, high availability, security, performance features, etc. It also provides better support for IA-64 platforms like HP-UX. With all the added features, Oracle9*i* Release 2 still remains a software product which can be optimized depending upon the platform-specific features and capabilities. This paper will discuss the new features in Oracle9*i* Release 2 that are actually advantageous for development and performance of database applications on HP-UX. Examples of revolutionary extensions in Oracle9*i* Release 2 for HP-UX, are Oracle9*i* Real Application Clusters using the HP Hyper Messaging Protocol (HMP) over HyperFabric II; enhanced high availability features and support for the most recent developments in the IA-64 architecture. Discussion on HP-specific enhancements for Oracle9*i* Release 2 will also include modifications to the existing implementations of HP Lightweight Timer and HP SCHED_NOAGE process scheduling policy. It will also provide a quick overview of other existing performance features and HP-specific enhancements in Oracle9*i* database required for optimal performance, e.g., useful oracle optimizer hints and parameters, turning on asynchronous I/O through HP asynchronous driver, etc. It will discuss how to optimally configure Oracle9*i* Release 2 on HP-UX by granting proper system privileges, tuning kernel parameters and managing the HP-UX shared memory segments. It will also comment on some of the post-installation issues and their solutions, for example, tuning the large virtual memory data pages to meet application needs and using the new LD_PRELOAD feature of the HP loader/linker for loading shared libraries required by an application. It is a common knowledge that one technique for improving performance, in addition to operating system and database level tuning, is to create efficient machine code and optimal programs by optimizing at the compiler and linker levels. This paper will acknowledge the efficient use of HP C compiler/linker flags and profile based optimization techniques for Oracle9*i* Release 2 on HP-UX 11*i*. While analyzing the enhancements, this paper will also refer to some of the recent record-breaking Oracle/HP benchmarks. To summarize, this paper will discuss a direction or a fast track to Oracle9*i* database Release 2 in making HP-UX a more robust platform for efficient database applications.

1. INTRODUCTION

Performance, scalability, manageability and high availability are basic requirements for today's business-intelligence applications. Oracle9*i* Release 2 has added features and capabilities that extend the existing investments of Oracle9*i*, in mission-critical infrastructure. Some of the key developments are in the areas of Data Guard, Oracle9*i* Streams, automated memory and space management, partitioning techniques, OLAP, high availability, security, performance features, etc. These features are advantageous for performance and development of database applications on any operating

[*] ***HP World 2002: Los Angeles, California, USA***

system in general, including HP-UX. Some of these new features will be discussed in Section 2. The already existing features in the Oracle9*i* release since the version 9.0.1, which are quite critical for performance tuning, are tabulated in Section 3. For more details about different Oracle9*i* features, please refer to [18].

The ultimate performance depends on a well designed application running against an efficient database, best optimized according to the operating system features and capabilities. An important need for proper configuration and optimization of Oracle products on HP-UX will be discussed in Section 4. Section 5 will provide a recap of the existing HP-specific enhancements in Oracle9*i*. The new features and extensions in Oracle9*i* Release 2 specific to HP-UX like the IA-64 support, the use of HP HMP (Hyper Messaging protocol) for Oracle9*i* Real Application Clusters and the modifications to the implementations of HP lightweight timer and SCHED_NOAGE process scheduling policy, will be discussed in Section 6.

2. KEY DEVELOPMENTS IN ORACLE9*i* RELEASE 2

This section describes a few of the several enhancements in Oracle9*i* Release 2 in the areas of ***data protection and high availability***, ***information integration***, ***database management***, ***partitioning techniques*** and ***performance features***. These developments are to ensure better functionality and performance of Oracle9*i* Release 2 across all platforms, in general. It is therefore imperative for the HP/Oracle developers to be knowledgeable about them. Each of these development features is described in the following sub-sections. The enhancements in Oracle9*i* Release 2, specific to HP-UX, will be described in the later sections.

2.1. Oracle9i (9.2) Data Guard

The loss of critical data or an inability to access data for an extended period of time can be catastrophic to a company doing e-business. The goal of Oracle9*i* Data Guard is to maintain a copy of the Production or ***Primary database*** to protect against corruption, human errors and disasters. In the event of a failure of the primary database, a ***Standby database*** can be activated to act as the new Primary database.

The Oracle9*i* Data Guard configuration is comprised of a collection of loosely connected systems, consisting of a single Primary database and a number of Standby databases. When using a Standby database, the changes made to the Primary database during transactions are logged locally and additionally, the redo log data generated by the changes are sent to the Standby database. These changes are applied to the Standby database which runs in a managed recovery mode or a read-only mode - the concept of a Standby database ***physically*** equivalent to the Primary database. Such a Standby database is called a ***Physical Standby*** database.

Oracle9*i* Release 2 extends the concept of a Physical Standby database technology of Oracle9*i* to that of a ***Logical Standby*** database. A Logical Standby database has the same logical schema as the Primary database but may have different physical objects such as additional indexes and materialized views. Unlike the Physical Standby databases, Logical Standby databases may be available for reporting as well as for simultaneous application of redo logs. A Logical Standby database is SQL maintained. Here a redo record from the Primary is not directly applied to the Logical Standby using block media recovery but rather the transactions are mined from the Primary database logs and applied to the Standby using SQL (using ***Log Miner*** technology [12]). The advantages of a Logical Standby in Oracle9*i* Release 2 are as follows:

- The Standby can remain ***Open*** and can be used to run reporting applications;
- The Standby can have a different physical layout than Primary and may be specially optimized for running reporting applications more effectively;
- The Standby database may be of a different version of Oracle (has to be 9*i* version 9.2 and onwards) and may be on a different version of the same operating system;
- The SQL-maintained Standby database has a better tolerance to corrupt logs because it eliminates the opportunity for a corrupt redo block to physically corrupt the Standby;
- The SQL-maintained Standby requires no application changes at the Primary database. However, additional information often needs to be captured at the Primary site to fully identify columns or primary keys within the redo records.

The Standby databases in the Oracle9*i* Release 2 Data Guard configuration can be a mix of both Physical and Logical databases. The Oracle9*i* Release 2 Data Guard architecture has the following major components (Figure 1):

1. The ***Log writer (LGWR) process*** is responsible for redo log buffer management and writing the redo log buffer entries to a redo log file or disk. The LGWR writes a commit record when a user process commits a transaction, or when the redo log buffer is one-third full, or when a database writer (DBWR) process writes modified buffers to disk. In a Standby database context, LGWR can write both to online redo log files and optionally to one or more Standby databases.

2. The ***Archiver Process (ARCn)*** copies online redo logs to a designated storage device after a log switch has occurred and when the database is in *ARCHIVELOG* mode or automatic archiving is enabled. During an extended network disconnection, log changes destined for a Standby, are accumulated in Archived Log Files. On resumption, the Archiver sends the changes one file at a time to the Standby database to re-synchronize it with the Primary.

3. The ***Log Transport Services*** manages shipment of log information from the Primary to Standby databases (not shown in Figure 1).

4. The ***Remote File Server (RFS) process*** receives and acknowledges receipt of changes from Primary, updates the Standby control file and writes changes to the Standby or Archived Log Files at the Standby site as required.

5. The ***Managed Recovery process*** manages the application of log information from the Primary to a Physical Standby database using managed block media recovery.

6. The ***Log Standby Apply Services*** component manages the application of log information in the Archived Log Files received from the Primary, transforms transaction information back to SQL (using Log Miner and Oracle Streams) and applies the SQL to a Logical Standby database using a streamlined interface.

7. The ***Data Guard Broker*** consists of agent processes associated with each Data Guard site providing unified monitoring and management infrastructure for an entire Data Guard configuration. It provides a command line mode (*DGMGRL*) and a graphical Data Guard Manager for users to interact with the Data Guard configuration.

The Oracle9*i* Release 2 also extends the Oracle9*i* Data Guard with three high level data protection modes - ***Maximum Protection, Maximum Availability*** and ***Maximum Performance***. These three modes replace the *guaranteed*, *instant*, *rapid* and *delayed* modes of data protection available in Oracle9*i* Release 1. For more details on Data Guard features, new attributes and management, refer to [13].

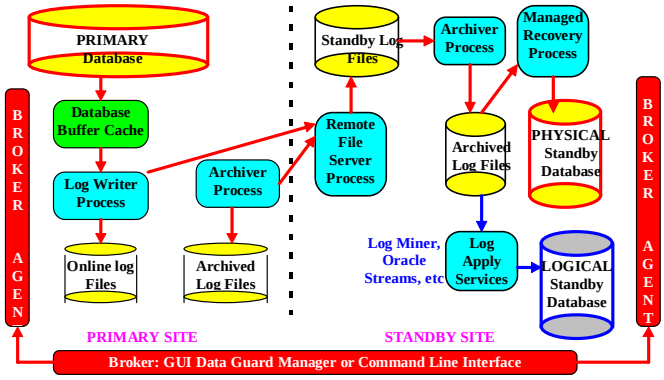

Figure 1: Oracle9*i* Release 2 Data Guard Architecture - a possible mix of Physical and Logical Standby databases, in addition to the Primary database.

2.2. Oracle9i (9.2) Streams

An important feature of a database management system is the ability to share information among multiple databases and applications. Traditionally, this has meant that users and applications must pull information from the database.

Today, however, new efficiencies and business models require a more comprehensive and automated approach. This active sharing of information includes capturing and managing events in the database, including and propagating those events to other databases and applications. Decision makers are often overwhelmed by the variety of options they face when selecting an information sharing solution and are in need of a single solution that meets all their information sharing needs.

Oracle9*i* Release 2 introduces a new information sharing feature, ***Oracle Streams***. Oracle Streams enables entire new classes of applications and in a single solution, satisfies the data movement, transaction propagation, and event management needs of most users. The propagation of data, transactions and events takes place in a data stream, either within a database or from one database to another. The stream routes published information to subscribed destinations.

Oracle Streams contains the following three basic stages:- ***a) Capturing Events; b) Staging and Propagating Events*** and ***c) Applying Events*** (Figure 2).

a) ***Capturing Events:*** Streams supports capture of events (database changes and application generated events) into a queue in two ways. ***Implicit capture*** enables the server to use a Capture Process (an Oracle background process) to capture *Data Manipulation Language (DML)* and *Data Definition Language (DDL*) changes from the redo log at the source database, format them into *Logical Change Records (LCRs)* and then enqueue them into a queue. ***Explicit capture*** allows user events which can be LCRs or user messages to be explicitly enqueued with a user application. The Capture Process can intelligently filter LCRs based upon defined rules so that only the specified types of changes to desired objects are captured.

b) ***Staging and Propagating Events:*** Streams uses queues to stage events for propagation or consumption between different queues in the same or different databases. LCR staging provides a holding area with security, as well as auditing and tracking of LCR data between the queues.

c) ***Applying Events:*** An Apply Process is an Oracle background process that dequeues events from a queue and either applies each event directly to a database object or passes the event as a parameter to a user-defined procedure called an apply handler. These apply handlers can include message handlers, DML handlers and DDL handlers. Oracle Streams includes a flexible apply process that enables a default or custom apply function. A custom apply sends an event to a user-created PL/SQL procedure for processing. This enables data to be transformed or formatted appropriately to meet the needs of a specific destination database. Support for explicit dequeue allows application developers to use Oracle Streams to notify applications, of changes to data, while still leveraging the change, capture and propagation features of Oracle Streams.

There can be one-to-many, many-to-one or many-to-many relationship between the source and destination queues - the queues being in the same or different databases. Figure 2 shows heterogeneous information sharing between an Oracle database with two destination databases which are respectively an Oracle and a non-Oracle database. For sharing with the non-Oracle database, the Oracle database functions as a proxy and the events intended for the non-Oracle database are dequeued in the Oracle database itself. An Apply Process at the Oracle database uses Heterogeneous services to apply the changes to the non-Oracle database across a network connection through a gateway. For more details on Oracle Streams, please refer to [27].

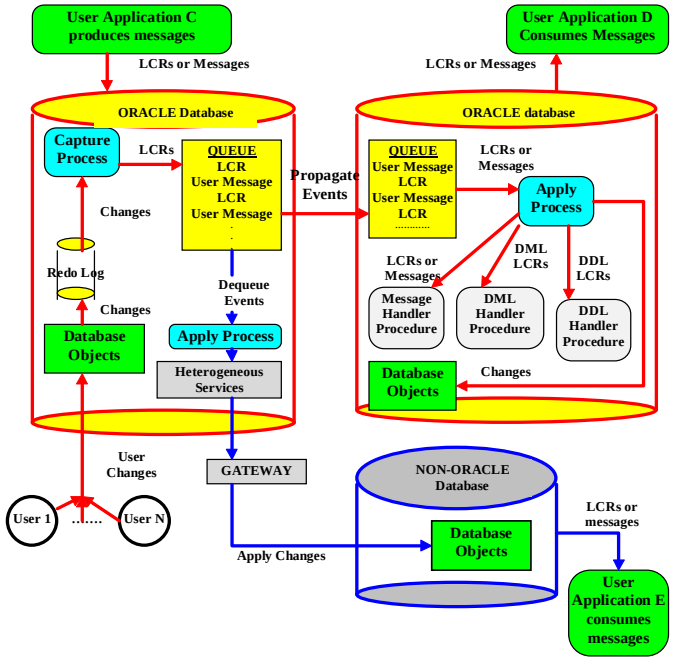

Figure 2: Heterogeneous data sharing using Oracle9*i* Streams.

2.3. Oracle9i (9.2) Real Application Clusters (RAC)

Oracle9i Real Application Clusters (RAC) is a cluster software architecture with scalability and high availability features harnessing the processing power of multiple interconnected computers. The Real Application Clusters (RAC) software and a collection of hardware (nodes or servers) known as a ***cluster***, unite the processing power of each component to create a robust computing environment.

2.3.1. Overview of the Oracle9i RAC architecture

Oracle9*i* RAC provides a single view of a database that is stored on many separate servers belonging to the same cluster. Data is stored on a shared disk and each server has access to a shared memory cache which is spread across all the servers in the cluster. It uses a cache-to-cache block transfer mechanism known as ***Cache Fusion*** to transfer read-consistent images of blocks from one instance to another, thus reducing disk access to a large extent. This is done by the ***Global Cache Service (GCS)*** and ***Global Enqueue Service (GES)*** integrated within the RAC, along with the operating system dependent (OSD) ***Cluster Manager (CM)*** and a high speed, low latency ***interconnect*** to satisfy remote requests for data blocks. The GCS and GES allow ***a) application transparency***, by providing a similar mechanism of shared resource access as in a single-instance Oracle9*i* database; ***b) fault tolerance***, by providing a distributed ***Global Resource Directory*** which is available across all active instances in the cluster, for the nodes to record information about resources; ***c) resource affinity***, by periodically nominating an instance as a resource master based on data access patterns; and ***d) data integrity***, by interacting with the OSD Cluster Manager for accurate status information of all instances. The different processes specific to a RAC database are the ***Global Cache Service Processes (LMSn)***, the ***Global Enqueue Service Monitor (LMON)***, the ***Global Enqueue Service Daemon (LMD)***, the ***Lock Process (LCK)*** and the ***Diagnosability Daemon (DIAG)*** (Figure 3).

2.3.1.1. Cache Fusion and its advantages

The synchronization of data across multiple caches in the nodes of a cluster is called ***cache coherency***. This is to ensure that reading a memory location through any cache will return the *most recent data* written to that location through any other cache. So it is also termed as ***cache consistency***. A diskless cache coherency mechanism in Oracle9*i* RAC that provides copies of blocks directly from a holding instance's memory cache to a requesting instance's memory cache is called ***Cache Fusion.***

The Cache Fusion addresses several types of concurrency like ***Concurrent Reads***, ***Concurrent Reads and Writes*** and ***Concurrent Writes*** on multiple nodes. An example scenario of Cache Fusion processing is when an instance (the requester) requests a data block from another instance (the holder), for modification purpose.

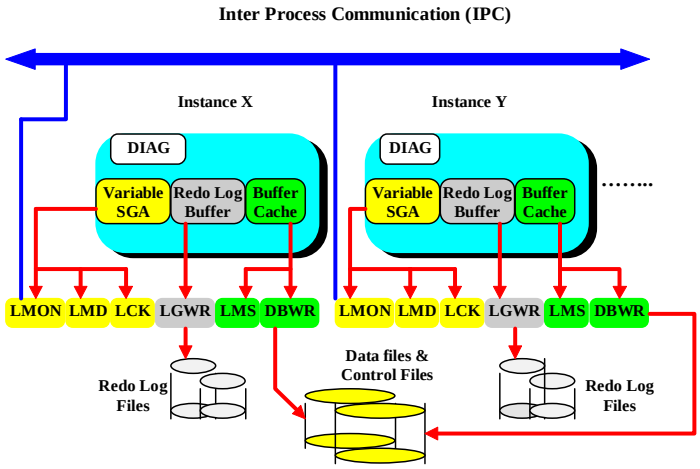

Figure 3: Oracle9*i* RAC-specific Instance Processes.

Figure 4 shows the requester posting such a request to the GCS. The holder keeps a copy of the requested block at a ***previous image state (PI)***, changes its mode from ***exclusive to null (X->N)*** and the role from ***local (L)*** to ***global (G)*** (as the block may later be changed by instances other than the requester). It then sends the block to the requesting instance along with the message that it has a ***PI*** of the block in role ***N*** and that the block on receipt needs to be changed to ***exclusive mode (N->X)*** with a ***global (G)*** role. The recipient follows the rule and sends the confirmation to GCS. It is important to note here that the data block has not been written to disk before the resource is granted to the requester. As a result, the performance overhead to manage shared data between instances is greatly diminished in this case (Figure 4). For more details on RAC, Cache Fusion and its advantages, please refer to [21].

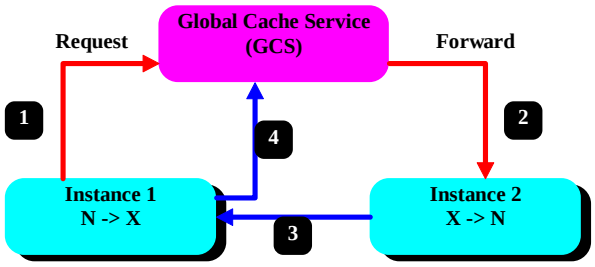

Figure 4: Cache Fusion: Requesting a changed block for a modification operation.

2.3.2. Oracle 9i (9.2) RAC Guard I

Oracle9*i* Release 2 extends the technology of RAC with the enhanced ***Oracle9i Real Application Clusters (RAC) Guard I*** along with the introduction of ***Oracle9i Real Application Clusters Guard II***.

The ***RAC Guard I*** formerly the RAC Guard, is an integral component of the Oracle9*i* Real Application Clusters software. It works with Real Application Clusters and the port-specific ***Cluster Manager*** to monitor and maintain availability of a Primary/Secondary cluster-node configuration. Its components (Figure 5) are the ***a) RAC Guard Packs*** which control the Oracle instances, IP addresses, monitors and listeners on each node; the ***b) PFSCTL control utility*** which controls the RAC Guard operations through a command-line user interface; the ***c) RAC Guard monitors*** detecting termination of an instance or unavailability of services and initiating failover; the ***d) RAC Guard Configuration template*** providing configuration templates for ease of RAC Guard configuration (not shown in Figure 5); and ***e) PFSSETUP Utility*** helping in automatic generation of necessary RAC Guard files with correct values, derived from the customized templates (not shown in Figure 5).

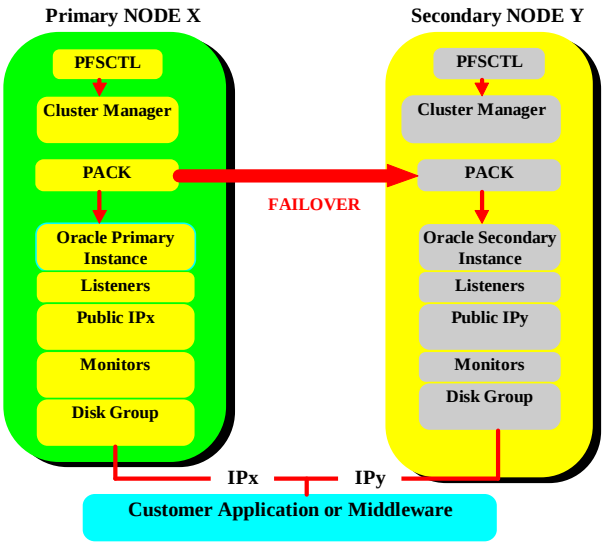

Figure 5: Oracle9*i* (9.2) RAC Guard I Architecture for a two-node cluster.

During failovers, there are some automatic actions that the RAC Guard I undertakes along with user-prompted commands, in order to get the desired final outcome. A possible failover scenario in a two-node cluster may be either of the three: a) the Primary instance fails; b) the Secondary instance fails and c) both the Primary and Secondary instances fail. Figure 6 shows how the first case is handled automatically by the RAC Guard I, in cooperation with the user and how the instances are restored after the failover. ***Step 1*** shows both nodes A and B operating normally with Packs A and B respectively running on them. The Packs A and B contain the IP addresses and the Primary and Secondary instances for the nodes A and B respectively. ***Step 2*** shows the failure of the Primary instance. In ***Step 3***, the Secondary instance becomes the Primary instance. Pack A starts on Node B in foreign mode. This means that only its relocatable IP address is configured to be up on Node B. ***Step 4*** shows how the user can restore the secondary instance role to node A by a ***restore*** command. The RAC Guard I starts Pack A on Node A with the Secondary instance role, because Pack B on Node B has already the Primary instance role due to failover. ***Step 5*** shows how the user can restore the original instance roles. With a ***move_primary*** command by the user, the RAC Guard I halts Pack B and the Secondary instance running on Node A becomes the Primary instance. When the user enters the ***restore*** command, the RAC Guard I starts Pack B on Node B with the Secondary instance role. ***Step 6*** shows the packs A and B now running on their home nodes with their original roles.

The RAC Guard I architecture is thus designed to build on the strengths of traditional high-availability solutions and provides the following functionalities:

- Automatic fast recovery and bounded recovery time from instance failures;
- Automatic capture of diagnostic data for certain types of failure;
- Enforced primary and secondary configurations;
- No delay when reestablishing connections after a failure.

For further details on RAC Guard I, please refer to [22].

2.3.3. Oracle9i (9.2) RAC Guard II

The Primary/Secondary instance configuration in RAC Guard I is the least complicated type of high availability configuration to configure and administer.

The administrative overhead for a Primary database in this configuration is the same as the overhead of a single-instance configuration. In such configurations, the secondary instance does not have to remain idle but can be available for read-only operations. Even though this a good high availability solution, it does not ensure exceptional scalability with growing clusters.

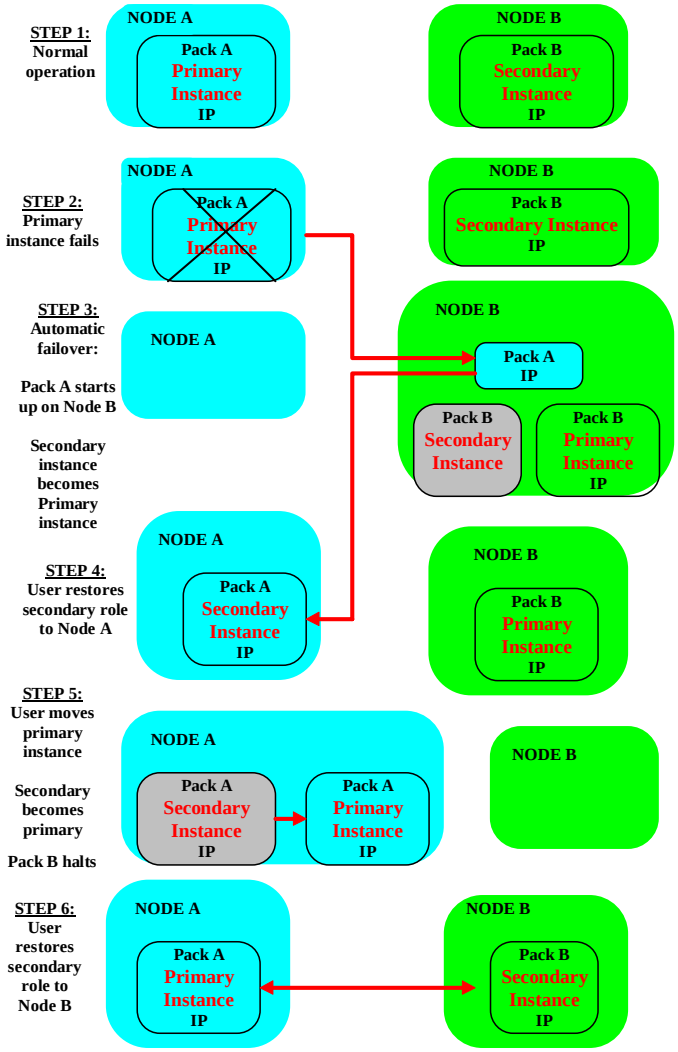

Figure 6: Oracle9*i* (9.2) RAC Guard I Operation.

Real Application Clusters Guard II in Oracle9*i* Release 2 extends the notion of a two-node (Primary/secondary) active/active cluster to an ***n-node, fully active cluster*** where all instances can support the services in the cluster database. Active/active instance configurations have typically been complex to configure. With the advent of Real Application Clusters Guard II, easy to manage full active configurations are available. It supports comprehensive ***workload management*** to maintain high availability for RAC databases and their applications. RAC Guard II transfers application loads based on the concept of ***service names***. Service names have been adopted for high availability because one does not have to make application changes to implement them. In addition, service names provide location transparency to the database instances that offer the service. Service names enable a single-system image that simplifies the configuration, operation, and recovery of workloads. RAC Guard II, therefore supports workload management based on service levels and ensures high availability for applications using database services [23].

2.4. Automated space and memory management

Automated space and memory management along with ***automatic undo segment management*** have been introduced as some of the new self-management features in Oracle9*i*.

To further simplify the ***space management*** at the database level, Oracle9*i* Release 2 allows all tablespaces including the ***SYSTEM*** tablespace to be ***Locally Managed*** thus simplifying the task of the database administrator. This can be done by specifying the ***EXTENT MANAGEMENT LOCAL*** clause in the ***CREATE DATABASE*** statement. Locally managed tablespaces provide better performance and greater ease of management over dictionary-managed tablespaces [11, 12]. The conditions that need to be met for this are :

- There should be a default temporary tablespace other than the SYSTEM tablespace;
- Rollback segments must not be created in dictionary-managed tablespaces but rather one should use automatic undo management;
- One cannot create any dictionary-managed tablespaces in the database or migrate a locally-managed tablespace to a dictionary-managed tablespace. Migration of an existing dictionary-managed SYSTEM tablespace to a locally-managed tablespace is allowed by the use of the ***DBMS_SPACE_ADMIN*** package.

Additionally, in Oracle9*i* Release 2, the Recovery Manager (RMAN) has a better space management feature while restoring the archived redo logs. The ***MAXSIZE*** option of the ***RECOVER <object>***

DELETE ARCHIVELOG command limits how much disk space RMAN may use when restoring logs during media recovery. This enables improved space management of archived log files thus freeing the DBA from managing the space allocation of the archived logs [24].

In the area of ***automated memory management***, Oracle9*i* Release 2 frees the DBA from time-consuming tuning and diagnostic tasks, by providing built-in ***advisories***. For example, the ***Shared Pool Advisory*** shows shared pool usage for improving parse time with minimum CPU resources and provides information on the optimal size of the library cache on the system. The ***PGA Advisory*** predicts the result of altering the PGA memory on the overall instance performance. Also see Section 2.6 and for more details on memory management, please refer to [20].

2.5. Enhancements in Partitioning Techniques

Partitioning capabilities in Oracle9*i* Release 2 have been expanded to support ***composite range-list partitioning***. This makes it much easier to perform operations on a list of partitions, by partitioning first by a ***range value***, say, a month, with a sub-partition on the ***list value*** (Figure 7). It is advantageous for data maintenance operations, for example, while doing backups of data for geographic regions by month. Furthermore, list partitioning now supports the concept of a default partition, so that, if a data row does not conform to the designated list of values, then the data row can be placed in a default partition instead of being rejected and generating an error. This means that applications no longer need to contain code to handle exception cases [11, 12].

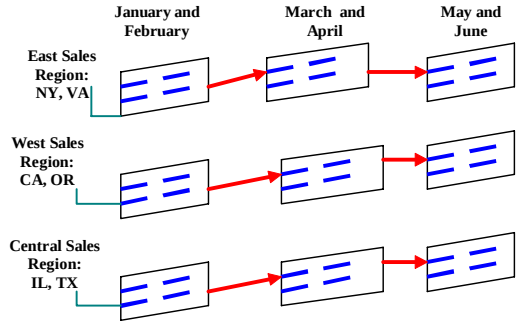

Figure 7: Composite Range-List Partitioning by range-value and list-value.

Additionally, parallel ***Data Manipulation Language (DML)*** is now supported on non-partitioned data tables, greatly enhancing the performance of a large update operation [14].

2.6. Enhancements in Performance features

Oracle9*i* Release 2 has the capability of ***dynamic sampling*** of optimizer statistics if a new initialization parameter ***OPTIMIZER_DYNAMIC_SAMPLING*** is set to a certain level between 0 and 10. This feature, if enabled, will dynamically gather statistics if the existing statistics are incomplete or known to be inaccurate. Dynamic sampling of optimizer statistics thus helps to improve performance by improving the quality of the statistics used by the query optimizer.

The new "out-of-the-box" ***Performance Tuning Intelligent Advisories*** in Oracle9*i* Release 2 allow the administrator to simulate a variety of hypothetical scenarios. These advisories use minimal resources and are available though the standard SQL interface.

- The ***Shared Pool Advisory*** statistics help to improve parse time and to minimize CPU usage. They track the memory usage incurred by SQL executions and helps shorten SQL execution time and minimize unnecessary CPU and I/O usage.
- The ***Mean-Time-To-Recover (MTTR) Advisory*** makes it possible for the administrator to set the time requirements to recover from a system crash without jeopardizing run-time performance.
- The ***PGA Aggregate Target Advisory*** makes it possible for the server to dynamically control the amount of PGA memory allotted to SQL work areas according to the ***PGA_AGGREGATE_TARGET*** limit set by the DBA.

For more details on Oracle9*i* Release 2 performance features, please refer to [20].

3. A RECAP OF THE EXISTING FEATURES IN ORACLE9*i*

This section provides a recapitulation of the performance features that exist since Oracle9*i* Release version 9.0.1. Table 1 provides a list of the performance features that help in tuning applications on any operating system, e.g., HP-UX 11*i*. All these generic features as

well as their tuning capabilities have been extensively discussed in [31]. The performance enhancements in Oracle9*i* Releases 1 and 2 specific to HP-UX, will be discussed respectively in Sections 5 and 6.

TABLE 1: Performance Features in Oracle 9*i* version 9.0.1 and beyond

Oracle9*i* Performance Features	Purpose
FIRST_ROWS_*n* Optimization	With the initialization parameter **OPTIMIZER_MODE** set 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 the first *n* number of rows (where *n* can equal 1, 10, 100, or 1000).
Literal Replacement with bind variables **CURSOR_SHARING parameter**	When bind variables are used in a SQL statement, the cost-based optimizer assumes that SQL cursor sharing is intended and that different invocations of the cursor are supposed to use the same execution plan. This helps effective sharing of the SQL area in the library cache. 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	With the **ALTER INDEX MONITORING USAGE** functionality, one can find the indexes that are not being used over a period of time and may drop them if necessary, thus reducing the cost of index maintenance.
System Statistics	For each plan generated by the query optimizer, estimates for I/O and CPU costs are also computed. This helps the optimizer 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	The parameter helps to 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.
Dynamic Memory Management	With release 9*i*, it is now possible to dynamically adjust the shared memory allocated for Oracle Shared Global Area (SGA) and the process-private memory allocated for the Process Global Area (PGA). There is an automatic mode to dynamically adjust the size of the tunable portion of the SGA and PGA memory by an instance. The sizes of the tunable portions are respectively adjusted based on the SGA_MAX_SIZE limit and an overall PGA memory target explicitly set by the DBA.

4. CONFIGURING AND OPTIMIZING ORACLE9*i* RELEASE 2 ON HP-UX

As both functionality and performance of Oracle RDBMS are sensitive to the operating system kernel configuration, compiler optimization and efficient use of operating system libraries, it is highly recommended to abide by all the installation and configuration recommendations in order to get the maximal performance of Oracle on HP-UX. This section describes the steps for configuring and optimizing Oracle products on HP-UX.

4.1. Configuring Oracle9i Release 2 on HP-UX

The steps to configure Oracle9*i* Release 2 on HP-UX are almost the same as those for Oracle9*i* Release 1 except for a few changes. A similar section in [31] discusses all details that one needs to be aware of while trying to configure Oracle9*i* on HP-UX. These are the ***a) required system privileges;*** the ***b) recommended HP-UX***

kernel parameter settings; and the ***c) recommended operating system patches, patch bundles*** and ***compiler/linker versions***. Unlike Release 1, it is not necessary to disable Data Prefetch [31] while running certain applications with Oracle9*i* Release 2 on HP Superdome (HP-UX 11*i*) due to the latch alignment changes in the later.

4.1.1. Oracle9i Release 2 on HP-UX: Pre-installation Recommendations

The ***system privileges*** required for Oracle9*i* to perform asynchronous I/O and use HP SCHED_NOAGE process scheduling policy, are ***MLOCK, RTSCHED*** and ***RTPRIO***.

Certain ***kernel operating system parameters*** e.g., ***shmmax, maxdsiz_64bit,*** etc, can be configured to fit specific system needs, resulting in a better application performance by an effective allocation of system resources. For example, the SHMMAX kernel parameter setting is critical for regulating the allocation of HP-UX shared memory segments for a 64-bit Oracle Instance. When a 64-bit Oracle server creates a database instance, the server creates memory segments by dividing the available shared memory by the value of the HP-UX SHMMAX kernel parameter. For example, if 64 GB of shared memory is available for a single Oracle instance and the value of the SHMMAX parameter is 1 GB, the Oracle server creates 64 shared memory segments for that instance. It is recommended to set the SHMMAX parameter value to the amount of available physical memory on the system. Doing this, ensures that the entire shared memory for a single Oracle instance is assigned to one shared memory segment and the instance needs only one protection key (each shared memory segment for an Oracle instance receives a unique protection key of the PA-RISC processor). To display the list of active shared memory segments on the system, enter the following command,

ipcs -m

If the Oracle server creates more than six segments for the instance, one should increase the value of the SHMMAX kernel parameter. Please refer to Section 7.1.4 for its performance implications. On HP-UX 11*i*, the kernel parameters are dynamically re-configurable. That means, users may dynamically alter parameters related to process memory, shared memory, etc, without going through a system reboot. Please refer to [17] for more information on recommended HP-UX kernel parameter settings for Oracle9*i* Release 2.

One also needs to make sure of installing the recommended versions of ***operating system patches, patch bundles, compiler/linker,*** etc. This is to ensure stability and performance of Oracle9*i* Release 2 on HP-UX. As the recommendations for the above steps are applicable to both HP-UX 11.0 and 11*i*, I have used the term HP-UX to cover both. For more details on each of the above steps as well as for the modifications made in Oracle9*i* Release 2, please refer to [31] and [17].

4.1.2. Oracle9i Release 2 on HP-UX: Post-installation Recommendations

This section will cover some of the post-installation issues with Oracle9*i* Release 2 (9.2) on HP-UX. Special scenarios related to allocation of process memory and dynamic loading of HP-UX shared libraries by applications, will be discussed here.

4.1.2.1. Large Memory Allocations and Oracle9i (9.2) tuning on HP-UX

One may see a significant increase in memory allocation while running applications with an Oracle9*i* Release 2 (9.2) executable, compared to memory allocation with an Oracle8*i* executable on HP-UX. There are two potential causes for this issue:

- The Oracle initialization parameter ***CURSOR_SPACE_FOR_TIME*** is changed from the default value ***FALSE*** to ***TRUE***.
- Oracle9*i* Release 2 (9.2) changes the default setting for virtual memory data pages from ***D (4KB)*** to ***L (1GB)*** on HP-UX..

Persistent Private SQL Areas and Memory: When a user submits a SQL statement to Oracle to be processed, Oracle automatically performs the following memory allocation steps:

1. Oracle checks the shared pool present in the Oracle SGA (Shared Global Area) to see if a shared SQL area already exists for an identical statement. If a shared SQL area exists, then Oracle uses that area to execute subsequent new instances of the statement. If a shared SQL area does not exist, then Oracle allocates a new shared SQL area in the shared pool for the SQL statement.
2. Oracle also allocates a private SQL area on behalf of the user session.

Private SQL areas contain data such as bind information and runtime memory structures for processed SQL statements. Private SQL areas are also where a parsed statement and other information for statement processing are kept. A cursor is a handle or name for a private SQL area; the cursor indicates that the private SQL area associated with the cursor remains in use.

Each user that submits the same SQL statement has a cursor that uses a single shared SQL area. Thus many private SQL areas can be associated with the same shared SQL area. If a user session is connected through a dedicated server, then private SQL areas are located in the server process PGA (Process Global Area). However, if a session is connected through a shared server, part of the private SQL area is kept in the SGA.

The Oracle initialization parameter *CURSOR_SPACE_FOR_TIME* specifies whether a SQL cursor can be deallocated from the library cache to make room for a new SQL statement. When this parameter is set to *TRUE*, then Oracle9*i* Release 2 (9.2) can only deallocate a shared SQL area from an Oracle library cache when all application cursors associated with the SQL statement are closed. Setting the parameter to *TRUE* also prevents the deallocation of private SQL areas associated with open cursors, thus making the user's private SQL area persistent. For more details, see [20].

Setting the *CURSOR_SPACE_FOR_TIME* initialization parameter to *TRUE*, accelerates SQL execution calls and improves performance, but leads to larger memory allocations for Oracle9*i* shadow processes due to an increase in cursor memory. On the other hand, setting *CURSOR_SPACE_FOR_TIME* parameter to *FALSE* degrades overall SQL execution and performance. See performance implications in Section 7.1.7.

Default Large Virtual Memory Page Size: By default, Oracle9*i* Release 2 (9.2) executable uses the largest virtual memory page size setting allowable by HP-UX, for allocating process-private memory. It is designated by the value ***"L" (largest)*** and is currently 1 Gigabyte on HP-UX 11*i*. This value is set as one of the ***LARGE_PAGE_FLAGS*** options, while linking an oracle executable. With the setting as *L*, the HP-UX operating system allocates the available process-private memory to pages of 1M, 4M, 16M, 32MB, 64MB and so on, until it reaches the 1 GB limit or it reaches the total amount of memory available for allocation. If one allocates enough memory to the Oracle PGA for the OS to be able to allocate memory in larger data page size units, then the OS allocates the maximum size at once. For example, if one allocates 48MB for the Oracle PGA, or a series of pages in unit sizes with the smaller multiples -- 4 (1MB) pages, 3 (4MB) pages and 2 (16MB) pages. If one allocates 64MB to the PGA, then the OS will allocate one page of 64MB, as the data page unit size multiple matches the available memory.

Large pages yield better performance, but if applications are constrained in memory and tend to run a very large number of processes, then this drastic page size increase may lead processes to indicate large memory allocations, followed by an "out of memory" error. In those cases, one must lower the page size to a value between the ***"D" (default)*** size of *4KB* and the ***"L" (largest)*** size of *1GB*. For example, an application may show reasonable performance with a *4MB* virtual memory page size setting. See performance implications in Section 7.1.7.

Tuning Recommendations: The following steps are recommended to address tuning for the increased memory allocation required for Persistent Private SQL Areas and Large Virtual Memory Page Sizes:

- Keep the value for CURSOR_SPACE_FOR_TIME to TRUE, unless the system indicates library cache misses while running the application. In that case, the shared pool may be small enough to hold the SQL areas for all concurrent open cursors.
- Decrease the virtual memory data page size for the Oracle9*i* Release 2 (9.2) executable. The page size setting can be altered by using the following command:

  /usr/bin/chatr +pd <new size> $ORACLE_HOME/bin/oracle

where the variable *newsize* represents the new value of the virtual memory page size. One can display the new setting using the ***chatr*** command as follows:

/usr/bin/chatr $ORACLE_HOME/bin/oracle

4.1.2.2. Using HP LD_PRELOAD environment variable for loading shared libraries

Because shared libraries require less memory, many programs use shared libraries. In most cases, the ***dld.sl*** 64-bit HP-UX dynamic loader is invoked automatically when applications using shared libraries start. At run time, the dynamic loader implicitly attaches to the process, all the shared libraries linked with the program. This includes the HP-UX ***thread-local storage***

(TLS) libraries. Programs can also use the ***shl_load()*** HP-UX function to:

- Explicitly access the 64-bit HP-UX dynamic loader;
- Attach a shared library to a process at run time;
- Calculate the addresses of symbols defined within shared libraries;
- Detach the library when finished.

Laboratory tests indicate that an error occurs when an application uses the ***shl_load()*** function to attach a shared library that directly or indirectly contains HP-UX TLS libraries. This error includes Oracle shared libraries, for example, ***libclntsh.sl*** which is currently linked with the ***libpthread.sl*** and ***libcl.sl*** HP-UX TLS libraries.

In the following example, the ***prog.c*** program calls the ***shl_load()*** function to load the ***libclntsh.sl*** library:

```
shl_load("<full path for
$ORACLE_HOME>/rdbms/lib/libclntsh.sl",
BIND_IMMMEDITE | BIND_VERBOSE |
DYNAMIC_PATH | 0L);
```

When the ***prog.out*** file is executed, it generates the following errors:

```
usr/lib/pa20_64/dld.sl: Cannot dlopen load
module '/usr/lib/pa20_64/libpthread.1' because
it contains thread specific data.

/usr/lib/pa20_64/dld.sl: Cannot dlopen load
module '/usr/lib/pa20_64/libcl.2' because it
contains thread specific data.
```

The new HP ***LD_PRELOAD*** environment variable resolves this problem. One can set the value of this variable to a colon-separated or whitespace-separated list of the TLS libraries that the dynamic loader can interpret. The dynamic loader treats the libraries specified by the *LD_PRELOAD* variable as the first ones in the link line and pre-loads these libraries implicitly at application startup. Therefore, calls to the ***shl_load()*** function do not return errors. For the example above, one should perform the following steps:

- Set the value of the *LD_PRELOAD* variable to include the paths of the TLS libraries that the program uses, for example:

```
export
LD_PRELOAD=/usr/lib/pa20_64/libpthread.1:
/usr/lib/pa20_64/libcl.2
```

- Enter the following command to execute the program, where *prog* is the name of the program:

```
prog.out
```

  No errors should appear.

- Unset the *LD_PRELOAD* variable to prevent unnecessary memory overhead by the loader:

```
unset LD_PRELOAD
```

For more information on the LD_PRELOAD environment variable, refer to HP documentation [5].

4.2. 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. One needs to use the desired level of optimization (1 through 4) and has to be aware of the trade-offs between the application code performance and the compile-time memory and CPU penalties. One may also want to activate optimization 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. The steps of PBO-ing an application and its advantages specific to Oracle products have been described in [31]. For more details on the method of PBO-ing, please refer to [5]. There is a method for ***dynamic instrumentation*** of an application using ***HP Caliper*** which is mainly used for optimizing executables on HP-UX Itanium Processor Family (see Section 6.2.1.2).

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 11.0/11*i*. The HP-UX system libraries also play an important role while linking a 64-bit optimized oracle binary on HP-UX 11.0/11*i*. For details regarding the use of the different flags/options and system libraries, please refer to the HP-UX reference manuals in [5].

5. A RECAP OF THE EXISTING HP-SPECIFIC FEATURES IN ORACLE9*i*

This section gives an overview of the various performance features, existing since Oracle9*i* Release version 9.0.1 on HP-UX. These enhancements include the HP-specific implementations of ***asynchronous flag*** in Oracle shared global area, the ***lightweight timer*** and the ***SCHED_NOAGE process scheduling policy***. The last two have been further enhanced in Oracle9*i* Release 2 and will be detailed in Section 6.

5.1. Asynchronous Flag in Shared Global area

Oracle9*i* 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 to it. HP requires that this flag be in shared memory.

Oracle9*i* configures an asynchronous flag in the SGA for each oracle process. Due to the polling facility provided by the implemented flag, Oracle9*i* on HP performs true asynchronous I/O where I/O requests can be issued immediately after the flag advertises that a bunch of I/Os have been completed. That is, Oracle9*i* can new I/O requests to the HP asynchronous driver, even though some of the previously submitted I/O operations are not complete. Please refer to Section 7.1.1 for performance implications.

5.2. HP Lightweight Timer for 64-bit Oracle 9*i*

Prior to Oracle9*i* 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. Oracle 9*i* on HP-UX uses a new library call function ***gethrtime()*** from HP to calculate elapsed time, thus reducing the negative impact on RDBMS performance when the *TIMED_STATISTICS* is set to *TRUE*. Further modifications on this topic for Oracle9*i* Release 2 has been discussed in Section 6.4. Also see Section 7.1.6 for comments on performance.

5.3. Using HP SCHED_NOAGE Process Scheduling Policy

By default, most processes run under the ***SCHED_TIMESHARE*** scheduling policy on HP-UX 11*i*. Here, each process has a priority and is given access to a CPU based on that priority. The priority degrades and the process is preempted by another process having a higher priority. In some cases, this 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, after which it realizes that it can't acquire the resource and goes 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.

Oracle9*i* uses a modified scheduling policy from HP, 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 suitable mainly for online transaction processing (OLTP) environments because OLTP environments can cause competition for critical resources. 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.

Further modifications on this topic for Oracle9*i* Release 2 has been discussed in Section 6.3. For more information on priority policies and priority ranges, one should also refer to the rtsched (1) and rtsched (2) man pages and the HP documentation site [5]. Also see Section 7.1.5 for comments on performance.

6. HP-SPECIFIC ENHANCEMENTS IN ORACLE9*i* RELEASE 2

This section describes all the HP-specific features and enhancements included in Oracle9*i* Release 2 on HP-UX. These are a) implementation of ***HP Hyper Messaging Protocol*** for Oracle9*i* Release 2 Real Application Clusters; b) implementation of ***Oracle9i Release 2 on IA-64 platform HP-UX***; c) modifications to the implementation of ***HP SCHED_NOAGE scheduling policy***; and d) extensions to the use of ***HP Lightweight timer.*** They have been discussed in sequence in the following sub-sections.

6.1. HP Hyper Messaging Protocol (HMP) for Oracle9i Real Application Clusters

For server clusters to be effective, they must provide large scalability and high availability. A shared disk cluster database architecture provides high availability but the efficiency of this approach depends on the inter-node communication mechanism involving direct memory access. Application programs, generally use the standard ***Transmission Control Protocol (TCP)*** and ***User Datagram Protocol (UDP)*** to transport messages to other programs across a network of computer systems. These protocols form the lightweight layer above the base ***Internet Protocol (IP)*** in the network layer. However, in a cluster configuration where several servers share the same disk, concurrent I/O operations can create contention and limit the cluster scalability.

HP and Oracle have collaborated over the past few years to provide a cluster solution that delivers high availability and scalability. HP has created the ***HyperFabric*** high-speed cluster interconnect network protocol for node to node communications and Oracle has developed the Cache Fusion technology in Oracle9*i* Real Application Clusters (RAC).

6.1.1. Advantages of the HP HyperFabric, HMP, MC/ServiceGuard and Logical Volume Manager

HyperFabric II (HyperFabric Release 2) is a high-speed cluster interconnect fabric, designed by HP, to meet the needs of enterprise-class parallel database applications. It provides ***higher speed*** (a link rate of 4 GB/s over fiber over a distance of 200 meters), ***lower network latency***, ***excellent scalability*** (up to 16 nodes via point-to-point connectivity and up to 64 nodes via fabric switches by transparent load balancing of connection traffic over multiple network interface cards (NIC)), ***greater reliability*** (by transparent failover of traffic from one card to another) and ***lower host CPU utilization*** compared to other industry standard protocols such as Fibre Channel. *HyperFabric II* supports both the industry standard *Transmission Control Protocol (TCP)* and *User Datagram Protocol (UDP)* over *Internet Protocols (IP)* and the HP proprietary ***Hyper Messaging Protocol (HMP).***

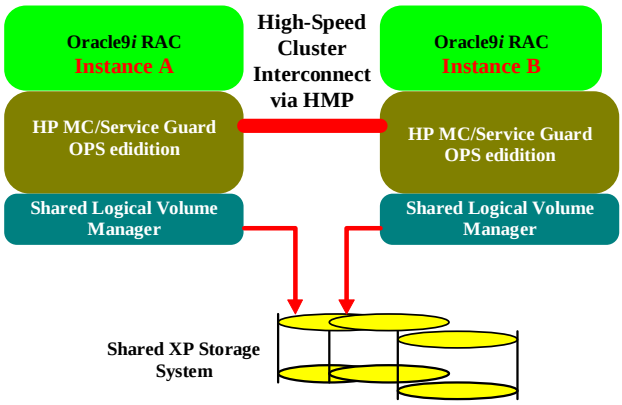

Figure 8: HP Hyper Messaging Protocol integration with Oracle9*i* RAC.

HMP expands the existing feature set of TCP and UDP by providing a true ***Reliable Datagram model*** for both remote direct memory access (RDMA) and traditional message semantics. HMP enables servers to directly access the memories and caches of other servers in the cluster and provides faster data transfer than UDP or TCP. With the operating system bypass capability and the hardware support for protocol off load provided by HyperFabric II, HMP provides ***high bandwidth***, ***low latency*** and extremely ***low CPU utilization*** with an interface and feature set, optimized for business critical parallel applications.

As described in Section 2.3, a major component of a cluster architecture is the port-specific ***Cluster Manager (CM)*** which monitors the heath of all nodes in the cluster and responds to failures in a way that enables outstanding level of availability for the applications. The Cluster Manager from HP, called the ***Multi-computer/ServiceGuard (MC/ServiceGuard) OPS edition***, is a special edition supporting Oracle9*i* RAC on

HP 9000 servers. Its major components are: a) the ***cluster manager*** which establishes and monitors various components of each node in the cluster; b) the ***distributed lock manager*** which enables reliable sharing of data between the nodes by enhanced lock management mechanism for coordinating and synchronizing the concurrent reads and writes to a database; c) the ***package manager*** which monitors and controls packages containing high available applications; d) the ***network manager*** which detects and recovers from card and cable failures. The *M/C ServiceGuard* (shown in Figure 8) has the ***ease of implementation and usage***, benefits of ***automatic application package failover*** and also the benefits of ***data integrity*** by coordination with the underlying database cluster components. For further details, see [5].

NOTE: HP's High Availability Product, formerly known as ServiceGuard OPS Edition, which is part of the product stack of Oracle9*i* Real Application Clusters (RAC) on HP, has recently gone through a product name change. From version A.11.14.01 onwards it will be called ***"ServiceGuard Extension for RAC"*** or in short, **SGeRAC.**

The shaped ***Logical Volume Manager (LVM)*** (shown in Figure 8) which ships with HP-UX, provides the basic functionality needed by the cluster nodes to share the physical disks and buses between themselves.

6.1.2. Advantages of Oracle9i (9.2) RAC

As described in Section 2.3.1.1, Oracle9*i* RAC ***Cache Fusion*** provides a single view of a database that is stored on many separate servers belonging to the same cluster. Data is stored on a shared disk and each server has access to a shared memory cache which is spread across all the servers in the cluster. Oracle9*i* Cache Fusion thus provides an ***expanded database cache*** for queries and updates with ***reduced disk I/O synchronization*** which overall speeds up database operations.

Additionally, the components of Oracle9*i* RAC such as ***RAC Guard I*** and ***RAC Guard II*** provide an efficient high availability solution. The enhanced coordination by the ***Global Cache Service (GCS)*** with the HP ***Cluster Manager (CM)*** called the ***M/C Service Guard OPS Edition*** leads to advantages like ***resource affinity***, ***data integrity***, ***application transparency*** and ***fault tolerance***, as already described in Section 2.3. The dynamic management of the nodes showing frequent access pattern for data blocks helps reducing communication latency by increasing the likelihood of local cache access. Overall, the Oracle9*i* (9.2) RAC with the use of ***minimal CPU resources*** and ***low communication latency*** is able to bring forth ***high availability*** and ***scalability*** without bandwidth constraints.

6.1.3. The Combined Advantages

Technologies like ***Cache Fusion, RAC Guard I, RAC Guard II*** and other enhanced recovery and high availability features of Oracle9*i* (9.2) RAC tightly integrate with ***HP HyperFabric*** technology to provide a) a robust cluster framework having large scalability without the need for data and workload partitioning and b) a configuration that leverages the best high-availability technology. Please refer to Section 2.3 and [5] for further details. The relevant performance data that proves the advantage of the combination is mentioned in Sections 7.1.2 and 7.2.3. Further work, combining RAC Guard II and HP HMP is currently being done, towards a more enhanced high availability solution foreseeable in future Oracle releases.

For more information about Oracle Real Application Clusters using HP Hyper Messaging Protocol (HMP), refer to [17] and [25].

6.2. Oracle9i Release 2 on the IA-64 (McKinley) platform, HP-UX 11.22

In general, the ***Reduced Instruction Set Computing (RISC)*** architecture provides a simpler instruction set and fewer components than the ***Complex Instruction Set Computing (CISC)*** architecture. As a result, RISC processors are smaller and run at higher clock speeds. However, the first RISC processors were unable to take advantage of the advancements in integrated circuit technologies due to an increased density of integrated circuits. To rectify this problem, the next generation of RISC processors were designed with ***multiple functional units*** that enabled simultaneous execution of multiple instructions in parallel. The task of the RISC processors to find valid parallel processing opportunities in a stream of instructions became more challenging with applications based on object-oriented programming languages with a hierarchical nature, for example, C++ and Java. The wide range of branching possibilities of these applications outran the ability of the RISC processor family to find parallel processing

opportunities and as a result they ran into a performance wall.

HP developed a concept of ***Explicitly Parallel Instruction Computing (EPIC)***, a cost-effective way to perform parallel processing where multiple possible branches of an application are executed simultaneously in parallel with the possibility of one branch being ultimately useful. The EPIC concept required:

- A new family of processors; and
- Compilers to generate efficient code for the new processors.

HP and Intel collaborated to create an architecture based on the EPIC architecture. The result has been the ***Itanium Processor Family (IPF)*** architecture - the Intel Corporation 64-bit microprocessor architecture or IA-64. Additionally, the IA-64 compilers from HP have been designed to generate efficient code for identifying parallel processing opportunities at compile time along with the ability of positioning retrievals of data from memory in advance of data requests by the application. The ultimate result has been a new model of HP-UX 11*i* (version 1.5 and beyond). With a very powerful processor underneath, it is capable of executing multiple tasks simultaneously and is able to achieve unprecedented levels of computational performance with large memory capabilities and vast amount of room to store, deliver and mine data [1].

6.2.1. The Combined Advantages

Just as the IPF based HP-UX 11*i* defines the standard for an industrial-strength enterprise operating system, designed and engineered to meet the high demands in computing power, Oracle9*i* Release 2 with features such as Cache Fusion, 64-bit large-memory addressing, fault tolerance and the capacity to manage very large workloads brings forth high performance, high availability and scalability to the Internet database arena for doing e-Business. The first Oracle9*i* release version 9.0.1.0 for the IA-64 platform HP-UX 11.20 (HP-UX 11*i* version 1.5) has been based on the ***Itanium*** chip. The goal for Oracle9*i* Release 2 on the IA-64 HP platform is to take advantage of the enhanced ***Itanium 2 (McKinley)*** based HP-UX 11.22 operating system. The primary areas around which Oracle9*i* Release 2 running on HP-UX 11*i* for IA-64 (McKinley) benefit most are described in the following subsections.

NOTE: As of the date of the presentation of this paper, Oracle9*i* Release 2 may not be available on the Itanium 2 based HP-UX 11.22 as scheduled. In that case, please look out for the availability of the release documentation [16]. It will provide additional information on installing and configuring Oracle9*i* Release 2 on Itanium 2.

6.2.1.1. Improved Compiler Optimization support on IPF

The HP compiler optimizer designed for IPF, employs a suite of transformations that take advantage of the key Itanium architectural features to improve instruction-level parallelism of Oracle applications. For example, the scheduler performs special techniques such as predication, control and data speculation. ***Predication*** allows control flow to be converted into conditionally-executed instructions that eliminates branch instructions and allows multiple execution paths to be executed simultaneously. It also enables ***software pipelining*** of loops in the program code by utilizing special branches for efficient scheduling. Lastly, ***speculation*** allows code to be executed earlier than it would be under the order specified by user. This helps loop-intensive numeric applications to achieve greater speedups on IA-64. Additionally, compiler optimizations at levels 3 (+O3) and above enable inlining of a *larger* set of math library routines into user code. The overall advantages are a) faster numerical codes which benefit more from the loop transformations; b) faster code for those which often call math library functions (sin, cos, sqrt, memcpy, memcmp, etc) because of their benefit from inlining. For more details, please refer to [1, 7, 33].

6.2.1.2. Improved Profile-based optimization support on IPF

Caliper-integrated dynamic instrumentation: As cited in Section 4.2, the Profile-based optimization (PBO) method is based on the static analysis of an application that is fed to the compiler. It requires recompilation with special flags, firstly, for inserting data collection code into the object program and secondly, for generating the final optimized code. ***HP Caliper***, an architecture for software developer tools dealing with binary programs uses the technology of ***dynamic instrumentation***, allowing program instructions to be changed on-the-fly with instrumentation probes. As opposed to static instrumentation, a) it is performed at run time of a program; b) it instruments only those parts of the program that are actually executed; c) it eliminates the overhead imposed by a separate instrumentation process involving recompiling; and also d) captures true run-time interactions with the processor and operating system,

e.g., a program's cache behavior and paging behavior. It integrates the Intel ***Itanium Performance Measurement Unit (PMU)*** support for enhanced sampling of over 150 event types allowing a wide range of system analysis tasks, e.g., analysis of cache misses, translation look-side buffer (TLB) or instruction cycles along with fast dynamic instrumentation.

Figure 9 shows the different components of the HP Caliper architecture: ***1) The Developer Tool Process*** physically spilt into two parts - the ***a) user interface***, which can be standalone scripts or integrated development environments and the ***b) HP Caliper shared object***, which contains support code, collectors, Caliper API and memory management routines; ***2) The Application Process*** which consists of the ***a) Application program*** and the ***b) HP Caliper Injected Object***. HP Caliper allows to inject an optional run-time library into the application process to record information, react on application events and communicate with the developer tool via the HP-UX debug interface ***(ttrace)***.

It is important to note that instrumentation with HP Caliper is supported only in combination with *+O1* optimization level of McKinley-based HP-UX compiler, unlike the one based on PA-RISC. For more details on HP Caliper, please refer to [2, 29].

Use of 64-bit profile counters: Static instrumentation with PA-RISC uses profile counters which are 32-bits in size. If the training run with instrumented executables is too lengthy, this may sometimes lead to counter saturation yielding wrong predictions. On McKinley, profile counters are 64 bits in size, thus allowing more lengthy training runs without concerns about counter saturation [7].

6.2.1.3. Desirable code scheduling support on IPF

Different Itanium-based implementations can have vastly different resource constraints, latencies and other scheduling criteria. The HP compiler optimizing scheduler for IPF, can currently schedule an application code for the first two Itanium Family implementations: ***Itanium*** and ***McKinley***. The user can schedule code to run best on each of these implementations by using the options ***+DSitanium*** and ***+DSmckinley***, respectively. Users might also want to optimize code once and have it run reasonably well on both implementations by using the option ***+DSblended***. One can also use the option ***+DSnative*** to schedule one's code to run fast for the implementation on which the code is compiled, whether it is *Itanium* or *McKinley*. See more details in [7]. The HP compiler support for Itanium-specific, McKinley-specific and blended scheduling models allows Oracle code to be scheduled at compile time in order to achieve the best performance on a particular implementation.

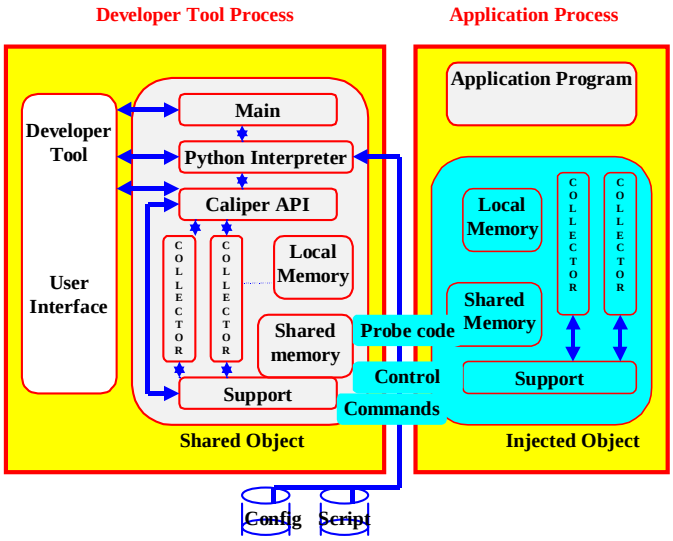

Figure 9: The HP Caliper Architecture for dynamic instrumentation.

6.2.1.4. Ease of compatibility and migration of Oracle9i Applications from PA-RISC

The existing 32-bit and 64-bit Oracle applications written for PA-RISC, as well as the 32-bit applications for Windows and Linux (if 64-bit clear) can run immediately on Itanium in ***compatibility mode*** without recompiling or recoding. This flexibility is a major advantage, especially for corporate developers who are responsible for large inventories of existing applications. While one gets effective performance in compatibility mode, one won't achieve the full power of IPF architecture without running one's applications in ***native mode***. This may not be an important consideration for non-performance critical applications, but it is strongly recommended that one ports other applications from compatibility mode to native mode at some point to take full advantage of the leading Itanium-edge capabilities [7]. This will require recompiling and may also require some recoding.

6.2.1.5. Enhanced compile-time performance tuning on IPF in native mode

The enhanced McKinley-based compiler provides up to two times better compile-time

performance tuning of C and C++ programs in native scheduled mode, compared to the Itanium-based compiler.

6.2.1.6. Combined responsiveness of IPF and Oracle9i Release2

The Itanium family architecture is designed to process massively parallel applications, has support for very large cache size (32KB L1, 256KB L2, 3MB L3 cache for McKinley), 64-bit memory addressibility and also ensures multiprocessor scalability. The McKinley chip, in particular, has a better support for branch instructions and concurrent load and store operations. Finally, the central core of McKinley runs at 1GHz, an advancement on the 800MHz core of Itanium (Figure 10). See more details on McKinley in [1, 3, 4, 8, 30].

Oracle9*i* Release 2 provides a significant new functionality in Online Analytical Processing (OLAP) and Data Mining by integrating the OLAP engine within the database server. The convergence of relational and multidimensional technology provides analytical capability with reduced information cycle time within the context of the Oracle database [19].

The above advantages enable customers running Oracle9*i*-based Web applications on IPF to perform queries at high speed and execute real-time data mining and smooth online transaction processing (OLTP) during peak periods.

6.2.1.7. Combined customization capabilities of IPF and Oracle9i Release 2

As businesses continue to integrate core applications into their e-Business environments, business data is also growing exponentially. The requirement of accelerated computations for data access, analysis and visualization leading to customization is becoming increasingly vital to the e-business arena.

The Itanium processor family (IPF) architecture provides ***computational*** and ***analytical efficiency*** due to its enhanced floating-point performance. It offers ***a) high precision basic Data type; b) Fused Multiply Add operation (FMA) unit; c) additional software for divide/Square-root operations;*** and ***d) a large number of floating point registers***. For a), Itanium's basic unified floating point format is 82-bit double extended which is an accurate and powerful format. The FMA unit referred in b), rounds up the multiply-add operations only once, can effectively double the floating point execution rate, can minimize or eliminate errors in misrounding and also can carry out integer multiplication using fixed-point multiply-add (XMA) instructions. For c), the traditional compute-intensive reciprocal and square root operations have been, respectively, replaced by fast instructions like FRCPA and FRSQRTA, thus allowing these operations to be carried out at once with better throughput for iterative loops. For d), the large set of 128 floating point registers makes it possible to pipeline significant floating point computations (Figure 10). It has instructions to move data between the floating point and general purpose registers without using memory for intermediate storage. Additionally, use of memory hierarchy is enhanced by allowing various forms of speculative executions thus reducing floating point latencies. These combine the HP compiler's in-line assembly capabilities to allow most of the detailed floating point architectural features in IPF to be directly accessed from C [1].

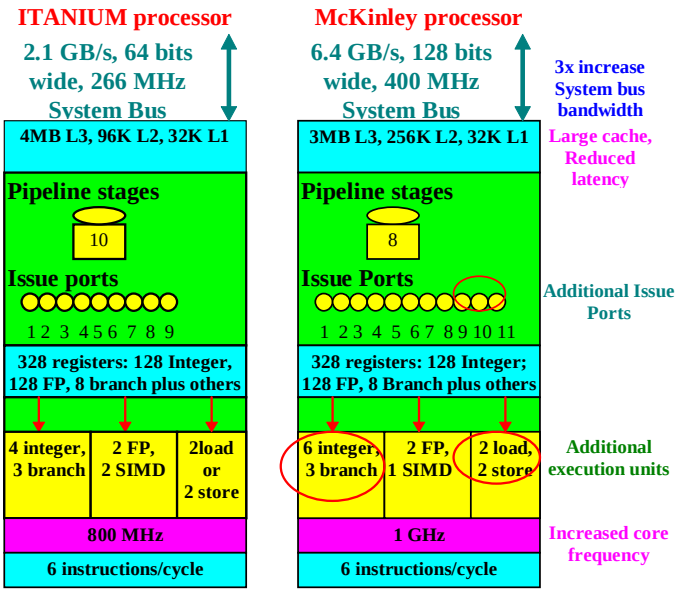

Figure 10: Itanium and McKinley IA-64 chip architecture.

All of the above further combine with the ***rich data type support*** in Oracle9*i* Release 2. Oracle9*i* Release 2 provides native support for ***XMLType*** data type within the database by providing a native, integrated XML database within the Oracle9*i* RDBMS. This a) enables better manageability of structured and unstructured data by users running their applications with different kinds of data, and b) provides enhanced access methods for navigating and querying XML [28]. Secondly, the ***Large Object (LOB)*** support in Oracle9*i* allows multiple users running multimedia applications to

simultaneously store, retrieve and manipulate different kinds of multimedia data [19]. Also, Oracle9*i* Streams in Release 2 enables propagation of heterogeneous data for efficient information sharing (Section 2.2).

The combined advantage of Oracle9*i* Release 2 and IPF is towards customizing the wide range of users with varying content needs. This includes high-performing ***data analysis***, faster execution of ***complex simulations***, and ***personalized web information service*** to e-business customers with varying user profiles.

6.2.1.8. Ensuring High availability

System error detection, containment, and recovery are critical elements of a highly reliable and fault tolerant computing environment. The degree to which this error handling is effective in maintaining system integrity depends upon coordination and cooperation between the system CPUs, platform hardware fabric and the system software. The machine check architecture model of IPF provides ***error containment*** as the highest priority, followed by ***error correction*** without program interruption, further followed by ***recording of error information***. System errors in IPF, may be handled by any of the following components: ***a) Processor hardware***; ***b) Platform hardware***; ***c) Processor firmware (Processor Abstract Layer or PAL)***; ***d) System Firmware (System Abstraction Layer or SAL)*** and ***e) Operating system (OS)*** (Figure 11). For hardware errors in (a) and (b), when the processor or the platform hardware corrects an error (e.g., in processor cache or system bus, etc), a notification of the corrected event is signaled to the OS through a ***Corrected Machine Check Interrupt (CMCI)*** for processor-corrected errors and through a ***Corrected Platform Error Interrupt (CPEI)*** for platform-corrected errors. When a processor or hardware detects an error that is not directly correctable by hardware (e.g., unmodified data in cache data structure, etc), a high priority ***Machine Check Abort (MCA)*** event is triggered and control is passed to the firmware, (c) and (d). The PAL and SAL firmware correct any error per their capability and control is returned back to the interrupted context. When an error cannot be corrected by the hardware or firmware layers, the control is transferred to the OS, (e). The OS corrects the error if possible and returns to the interrupted context or switches to a new context. This type of error handling model and possible self-correction and self-recovery mechanism ensure non-disruption of the running application, thus leading to high availability. See more details in [4, 6]. The above advantages combine with those of the Data Guard (covered in Section 2.1) and RAC (Section 2.3) components of Oracle9*i* Release 2 for creating a ***highly available***, and ***efficient computing*** environment for Oracle e-business customers.

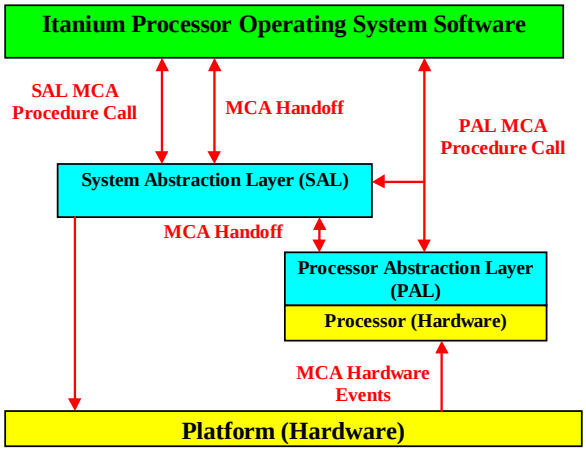

Figure 11: Itanium Processor Family (IPF) Error Handling Model.

6.2.1.9. Ensuring Security

Appropriate security is central to Oracle e-Business customers, both for external company Web sites and for internal company networks. Encryption algorithms like the ***private key*** and ***public key algorithms*** with proper scrambling of exchanged information lie at the heart of such security systems.

The IPF architecture has high-performing encryption/decryption capabilities due to ***a) its support for fully pipelined, 64-bit Integer Multiply-Add instructions*** for ***RSA***™ (public key operations) computations; ***b) the 128-bit register width***; ***c) large number of registers*** - 128 integer registers, 128 floating point registers and 8 branch and predicate registers for both Itanium and McKinley (Figure 10) and ***d) fast floating point performance***. Thus, the data commonly referenced by the encryption algorithms can be stored in the registers with a smaller amount of data to be accessed from memory. Also the large number of register sets allows the processor to hold inside the registers, all the partial products generated in a RSA thus contributing to the Secure Sockets Layer (SSL) communication speed up [3, 4].

™ ***RSA stands for RSA Security, Inc.***

Oracle9*i* Release 2 also provides a secure application development and deployment platform. It allows a username/password on a CREATE DATABASE statement and allows setting up a number of default accounts which are locked and expired upon installation, thus providing additional security. The database administrator can also grant or revoke object privileges on another user's objects. Additionally, there are *two* enhanced security options - ***a) Oracle Label Security***, which secures data releasibility by providing sophisticated methods of dissemination of data, for example, by using ***data labels*** like ***CONFIDENTIAL, SECRET*** or ***TOP SECRET*** and by using ***data security compartments*** like ***NATO*** and ***CRYPTO***; ***b) Oracle Advanced Security,*** which now supports the ***Advanced Encryption Standard (AES)***, accepts authorization from an industry-standard ***Remote Authentication Dial-In User Service (RADIUS)*** server and provides a User Migration Utility to migrate password-authenticated database users to Oracle Internal Directory for centralized management [26]. All these combine with the Oracle9*i* Release 2 database support for both public and private keys to deliver an added level of application and data security to the e-business customers.

To summarize, Oracle9*i* Release 2 on the IA-64 HP-UX platform provides customers with the scalability, availability, security and customization needed for e-business tomorrow. For comments on internal performance testing with Oracle9*i* Release 2 on IPF (McKinley), see Section 7.1.3.

6.3. Extensions to implementation of the HP SCHED_NOAGE policy

The ***SCHED_NOAGE*** policy has been implemented in Oracle9*i* Release version 9.0.1. Details of the use of the HP *SCHED_NOAGE* policy over the *SCHED_TIMESHARE* policy has been discussed in Section 5.3. There has been some extensions to this implementation in Oracle9*i* Release 2. To use the HP *SCHED_NOAGE* policy, it is mandatory that one sets the Oracle initialization parameter, *HPUX_SCHED_NOAGE* for each instance. The integral value of this parameter specifies process priority levels. On HP-UX 11.0, the range of this parameter is 153 to 255 and on HP-UX 11*i*, the range is 178 to 255. Prior to Oracle9*i* Release 2, if the user unknowingly sets the parameter to a value which is out of range for a specific OS version, Oracle9*i* processes will ignore the *SCHED_NOAGE* policy and continue with the default *SCHED_TIMESHARE* policy. Modifications to the implementation allows Oracle9*i* Release 2 to automatically set the parameter to a permissible value and continue with the *SCHED_NOAGE* policy using the newly set priority. It also generates a message in the alert_*sid*.log file about the new setting. However, it is strongly recommended to set the parameter to a desired priority level in order to schedule the Oracle processes with that priority.

6.4. Modifications to Lightweight Timer Implementation

As discussed in Section 5.2, setting the Oracle initialization parameter, ***TIMED_STATISTICS*** to *"TRUE"* at the instance level, directs the Oracle server to record the total elapsed time and wait time for significant wait events. This data is useful for comparing the total wait time for an event to the total elapsed time between performance data collections. This helps to tune any performance problems related to the wait events. In Oracle9*i* Release 2, timed statistics are ***automatically*** collected by the Oracle database if the dynamic initialization parameter ***STATISTICS_LEVEL*** is set to the default value, ***"TYPICAL"*** or it is set to ***"ALL"***. This is because, the default setting, "*TYPICAL*", implicitly turns on the *TIMED_STATISTICS* initialization parameter to "*TRUE*". If the user dynamically sets the initialization parameter, *STATISTICS_LEVEL* system-wide, to a value "***BASIC***" (using the ***"alter system"*** command), then the *TIMED_STATISTICS* parameter also sets itself dynamically to the value "*FALSE*". With the altered *STATISTICS_LEVEL* in place, one will have to explicitly set *TIMED_STATISTICS* to "*TRUE*", if one desires to collect timed statistics later on. The *TIMED_STATISTICS* parameter in Oracle9*i* Release 2 is dynamic. So one can use the appropriate "alter" statement to either turn it on or off, without shutting down the Oracle instance.

Due to the above changes in the parameter *TIMED_STATISTICS*, Oracle9*i* Release 2, unlike Oracle9*i* Release version 9.0.1 on HP-UX systems, uses the ***gethrtime()*** system library call by default to record the elapsed time. This is true as long as one explicitly does not change the *STATISTICS_LEVEL* parameter to "*BASIC*"or alters the *TIMED_STATISTICS* parameter to *"FALSE"*. The default settings enable one to collect run-time statistics at all times while running an Oracle instance.

For comments on performance implications, please refer to Section 7.1.6. Also for more details on the

modifications made to the *TIMED_STATISTICS* parameter, please refer to [20].

7. PERFORMANCE ANALYSIS

This section analyzes the various performance enhancements in Oracle9*i* Release 2 on HP-UX 11.0/11*i* that have been detailed in the earlier sections. Tests have been done internally at Oracle and HP performance labs and standard benchmarks have been carried out in order to measure the above enhancements.

The analysis has been divided into two parts:- Firstly, the results from laboratory testing will be discussed, where each of the enhancements has been measured separately on HP-UX 11*i*. Lastly, the published benchmark data will be quoted. These serve as good standards for measuring performance of different types of applications using Oracle9*i* Release 2 (9.2) on HP-UX 11*i*.

7.1. Laboratory Analysis

In this section, all the enhancements in Oracle9*i* Release 2 on HP-UX, will be analyzed in sequence. The performance implications related to some of the pre-installation and post-installation issues will also be discussed here. The laboratory data is based on medium-scale TPC-H and TPC-C (*TPC* stands for transaction Processing Council) baseline runs on HP-UX 11.0 and 11*i* after installing the proper operating system patches, patch bundles, compiler/linker and a few other patches related to the enhancements.

7.1.1. Asynchronous I/O for Oracle9i

The asynchronous flag implementation for Oracle9*i* Release version 9.0.1 and onwards, is described in Section 5.1. This implementation provides true asynchronous I/O functionality for Oracle9*i*. It helps to enhance the overall RDBMS performance and ensures good scalability of parallel I/O processes e.g., parallel query slaves, for very large Oracle databases. Lab tests show that this implementation provides up to 13% improvement in DSS performance. Also see Section 7.2.2 for DSS benchmark results.

7.1.2. Oracle9i RAC and HMP

The objective of a series of laboratory testing with Oracle9*i* RAC and HP HMP, has been to ensure high quality of both products and prove the scalability, high availability and resilience of large cluster configurations of up to 16 nodes of Oracle9*i* RAC. The test framework included a test driver which can be run against any cluster configuration and has the ease to setup, use, vary workload and carry out fault injection. A combination of the products like Oracle9*i* RAC, HP Service Guard OPS edition and HP HMP were installed and configured on a wide range of HP servers (A, L and N-classes). They had to undergo the following tests: ***a) Stress tests***, which involved a number of orderly and random start-ups of one or more Oracle database instances, addition and removal of workload and instance shutdowns. The workload has been an Online Transaction Processing (OLTP) workload executing statements like, *SELECT*, *INSERT*, *UPDATE*, *DELETE* and *VERIFY*; ***b) Recovery tests***, which involved testing block recovery, caused by the death of database instances, lost blocks and cancellation by CTRL-C; and ***c) Destructive tests***, which involved forced failures by *Oracle software* (with one or more background processes killed manually); by *OS software* (with one or more cluster daemons killed manually, or the system being forced to reboot); and by *hardware* (with network or disk connectivity or power supply manually removed).

The above tests showed successful results from stress tests as well as successful cluster reconfigurations during destructive test phase. Laboratory results with Oracle9*i* RAC using HMP cluster interconnect, indicate a 1.78 scalability factor moving from a one-node to a two-node HP N4000 cluster and a 1.9 scalability moving from a two-node to a four-node HP N4000 cluster. Laboratory tests with a 16-node HP cluster configuration, also show a 20% improvement in Oracle9*i* RAC OLTP throughput and response time using HMP over UDP.

7.1.3. Oracle9i Release 2 on HP-UX 11.22 for IA-64 (McKinley)

In general, McKinley performance has around 1.5x to 2x times better performance over Itanium [8] due to all the qualities described in Section 6.2 (Figure 12). As of the date of writing this paper, Oracle9*i* version 9.0.1 is readily available on HP-UX 11.20 for Itanium [15]. But, Oracle9*i* Release 2 is available only on PA-RISC and is scheduled to be available on IPF (Itanium 2

or McKinley), very shortly. So all data in this section are laboratory based and are subject to change as the release work goes on. Please refer to up-to-date information in the documentation [16], once it is made available.

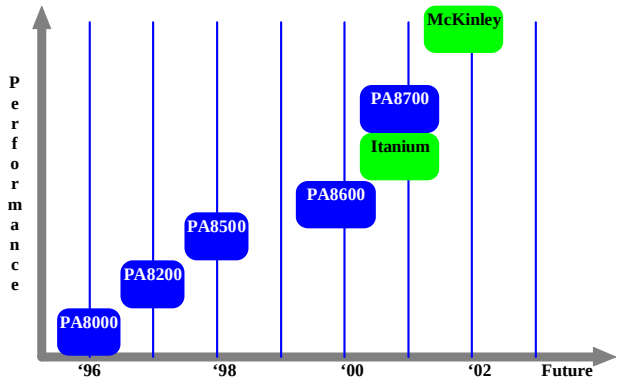

Figure 12: Performance of PA-RISC Processor family versus Itanium Processor Family.

Pre-release laboratory testing on a 4-way Itanium 2 (McKinley) based HP server running HP-UX 11.22 (McKinley-based OS version) indicate a 100% increase in Oracle9*i* Release 2 OLTP performance compared to that on HP-UX 11.20 (Itanium-based OS version).

7.1.4. HP-UX Kernel parameter SHMMAX and Oracle9i Release 2

As described in Section 4.1.1, it is strongly recommended to set the HP-UX kernel parameter *SHMMAX* to the available physical memory for 64-bit applications with Oracle. As Oracle9*i* is only 64-bit, the importance of this parameter is critical. In laboratory tests, performance degradation occurs when the 64-bit Oracle9*i* instance creates more than *six* shared memory segments. This is because, there are six protection keys available for shared memory segments on the PA-RISC processor, with a unique protection key for each shared memory segment. So, if one's system has more than six shared memory segments, the HP-UX operating system displays protection key faults and hence degrades performance. The recommendation is well documented for Oracle9*i* Release 2 [10].

7.1.5. HP SCHED_NOAGE Process Scheduling policy for Oracle9i Release 2

The implementation of HP process scheduling policy ***SCHED_NOAGE*** and its modifications have been discussed respectively in Sections 5.3 and 6.3. Laboratory tests show that Oracle9*i* OLTP performance increases by up to 10 percent using the HP *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. However, it is recommended to test and verify prior to use, whether the *SCHED_NOAGE* policy is of benefit to one's application environment.

Oracle9*i* Release 2 extends the implementation by generating alert messages in case the Oracle9*i* initialization parameter, *HPUX_SCHED_NOAGE* is out of range for a particular OS version. This is a very important and an useful addendum in the new release but does not incur an additional performance improvement.

7.1.6. HP Lightweight Timer for Oracle9i Release 2

Laboratory tests with Oracle9*i* Release version 9.0.1 (discussed in Section 5.2) show that with the initialization parameter ***TIMED_STATISTICS*** set to *"TRUE"*, the HP library call ***gethrtime()*** provides up to 10% performance improvement over the previous implementation.

The modifications to the implementation of HP Lightweight timer for Oracle9*i* Release 2 (described in Section 6.4), allows one to collect run-time statistics at all times, because the initialization parameter *TIMED_STATISTICS* is *"TRUE"*, by default. Laboratory tests show that the new implementation using the *gethrtime()* system library call by default, at all times, yields no negative impact on Oracle9*i* Release 2 OLTP performance, when compared to the implementation using *gethrtime()* system library call only when the *TIMED_STATISTICS* initialization parameter is set to *"TRUE"*.

7.1.7. Large Memory Allocations and Oracle9i Release 2

The possibility of seeing large memory allocations with Oracle9*i* Release 2 on HP-UX and corresponding tuning recommendations have been

described in Section 4.1.2.1. One reason is the Oracle initialization parameter, *CURSOR_SPACE_FOR_TIME* when set to the value *TRUE* and the other reason is the HP port-specific large virtual memory page size setting for the Oracle9*i* Release 2 executable.

The Oracle Initialization Parameter CURSOR_SPACE_FOR_TIME: Lab tests with Oracle Applications, show that setting the *CURSOR_SPACE_FOR_TIME* initialization parameter to *TRUE*, offers the following advantages:

- It accelerates SQL execution calls, because each active cursor's SQL area is present in memory and never aged out.
- It improves application performance, as Oracle9*i* Release 2 (9.2) is allowed to bypass the procedure to verify if a shared SQL area is in the library cache. By retaining private SQL areas between SQL statement executions, it also helps to save cursor allocation and initialization time.

Lab tests with Oracle Applications also show that setting the Oracle initialization parameter *CURSOR_SPACE_FOR_TIME* to *TRUE* in Oracle9*i* Release 2 (9.2) causes the following disadvantages:

- It increases the memory requirements of user processes due to an increased memory allocation for the persistent private SQL areas, in comparison to Oracle8*i*.
- It significantly increases cursor memory in comparison with Oracle8*i*, leading to larger memory allocations for Oracle9*i* shadow processes.

In lab tests, keeping or changing the value of *CURSOR_SPACE_FOR_TIME* parameter to *FALSE*, results in degraded overall SQL execution and performance because it results in rapid deallocation of shared SQL areas from the library cache. See tuning recommendations in Section 4.1.2.1.

Large Virtual Memory Pages: In general, large memory pages yield better application performance by reducing the number of virtual memory translation faults that need to be handled by the operating system. This makes available, more CPU resources for the application. Large pages help in reducing the total number of data pages needed to allocate the process-private memory, thus leading to a lower likelihood of Translation Lookaside buffer (TLB) misses at the processor level. However, for applications which are constrained in memory, the increased page size may result in paging and swapping and even "out of memory" error. The tradeoff for reducing the page size is a greater probability of TLB misses, higher CPU utilization and a performance overhead.

Lab tests show that with the lowest page size *(4KB)* setting, CPU utilization is as high as 20%. With the highest setting of *"L"*, the memory consumption is 50% higher than that with a *"4MB"* setting. So, in cases where the system shows memory constraints, it is recommended that one sets the page size appropriately for a particular type of application, within the constraints of available memory resources. See tuning recommendations in Section 4.1.2.1.

The Combined effect: Tests with Oracle Applications using Oracle9*i* RAC on HP-UX 11*i* show a reasonable performance and good scalability with a *4MB* virtual memory page size setting and with the Oracle initialization parameter *CURSOR_SPACE_FOR_TIME* set to the value *TRUE*.

7.2. Published benchmarks with Oracle9i on HP-UX 11i

The recently published benchmarks using Oracle9*i* on HP-UX 11*i*, are enlisted in this section. The cumulative advantages of Oracle9*i* and the HP-UX operating system as described throughout this paper, correlate to the benchmark results using very large configurations and also act as valid data points of their combined performance.

7.2.1. HP/Oracle Online Transaction Processing (OLTP) benchmark

Table 2 shows the system, hardware and software specifics for the published HP/Oracle ***TPC-C*** benchmark using Oracle9*i* Release 2 (9.2) on HP-UX 11*i*. For more details on this benchmark, please refer to [32].

7.2.2. HP/Oracle Decision Support Systems (DSS) benchmark

Table 3 shows the system, hardware and software specifics for the recently published HP/Oracle ***TPC-H*** benchmark using Oracle9*i* Release 2 (9.2) on HP-UX 11*i*. For more details on this benchmark as well as other benchmarks with Oracle9*i* on HP-UX 11*i*, please refer to [32].

Applications Standard benchmark 11.5.3 using ***Oracle9i Real Application Clusters.*** The table shows two benchmark configurations and the achieved performance and scalability results in a cluster configuration. These benchmarks have been run using Oracle9*i* RAC version 9.0.1 on HP-UX 11*i*. For more details on these benchmarks, please refer to [9].

TABLE 2: Oracle/HP TPC-C benchmark

TPC-C Benchmark	
Configuration Specifics	*64-way PA-8700 (750 MHz) Superdome (HP-UX 11i)*
	Oracle9i Database, Release 2 (9.2)
	308,000 users
	256 GB of memory
	25 PCI Fibre Channel adapters
	74 SureStore Virtual Array VA7100 with 420 ×18GB and 690 ×36GB, 15K RPM drives
	Total storage of 14607 GB
Performance	*389,434 transactions per minute (tpmC)*
	Price/Performance of $16.41 per transaction.

TABLE 3: Oracle/HP TPC-H benchmark

TPC-H Benchmark	
Configuration Specifics	*64-way PA-8700 (875 MHz) Superdome (HP-UX 11i)*
	Oracle9i Database, Release 2 (9.2)
	1000 GB database size
	128 GB of memory
	25 PCI Fibre Channel adapters
	1 HP SureStore disk system 2100 with 3 ×18GB Ultra 3 SCSI drives; *84 SureStore Virtual Array VA7100 with 1260 ×18GB, 15K RPM drives*
	Total storage of 22734 GB
Performance	*25,805.4 queries per hour (QphH@1000 GB)*
	Price/ Performance of $213.00 per QphH@1000GB

7.2.3. Oracle Applications 11i Standard Benchmark with Oracle9i RAC on HP-UX 11i

Table 4 shows the configuration details of the Application Tier and the Database Tier as well as the storage used for the publication of the HP/Oracle ***Oracle***

TABLE 4: HP/Oracle Oracle Applications Standard Benchmark 11.5.3

Oracle Applications Standard Benchmark Version 11.5.3						
Applications Tier	*8-way PA8600 (550 MHz) per cluster node* *32GB memory per cluster node* *HP-UX 11i* *Oracle 9i Application Server* *TCP over HyperFabric I interconnect to Database Tier* *2 ×18GB HP internal database disks*	*4-node cluster of HP rp7400 servers*	*User Count 2296*	*8-node cluster of HP rp7400 servers*	*User Count 4368*	*Scalability factor moving from a 2-node cluster to a to 4-node cluster is 1.95*
Database Tier	*4-way PA8600 (550 MHz) per cluster node* *16GB memory per cluster node* *HP-UX 11i* *HMP over HyperFabric II interconnect for Oracle9i RAC* *Oracle9i RDBMS Server* *2 ×36GB HP internal disks*	*2-node cluster of HP rp5470 servers*	*Average Response Time 1.16 secs*	*4-node cluster of HP rp5470 servers*	*Average Response Time 1.25 secs*	
Database Storage	*SureStore E: Disk array FC60, 5.36 TB disk storage shared among the two tiers*					

8. CONCLUSIONS

This paper covers the technical details of some of the major achievements of Oracle9*i* RDBMS on HP-UX. The focus has been mainly on the recent developments for Oracle9*i* Release 2 (9.2) on HP-UX and how the combined best features of both yield an optimal performance. As all in-depth technical details cannot be covered in this paper, it is strongly recommended to refer to the citations for more information. The immediate future direction is to work on several enhancements for the upcoming Oracle10*i* release on HP-UX 11*i*. One major area will be to combine Oracle 10*i* RAC with the most recent high availability products from HP as well as with the newer InfiniBand architecture models. Directions of further work will be towards implementing the Oracle10*i* Storage Manager technology on HP-UX and also in implementing the best features of the current and upcoming versions of HP-UX 11*i* into Oracle10*i* and beyond. As the technology of 64-bit Itanium Processor Family Architecture advances with the advent of Intel microprocessors like Madison, Deerfield, etc, and is simultaneously implemented and coerced with the HP-UX 11*i* architecture, it will be a task on its own for the future releases of Oracle RDBMS on HP-UX 11*i*, to get the best out of the advanced technology, especially in areas like compilers, operating system software and application optimizations - both static and dynamic.

9. REFERENCES

1. *Developer and Solution Partner Portal - Itanium Processor Family: Hewlett Packard.*
2. *Hundt, R. HP Caliper - An Architecture for Performance Analysis Tools. Paper published, October 2000.*
3. *Intel Developer Services: Intel Corporation.*
4. *Intel Itanium Architecture Software Developer's Manual, Volumes 1, 2 and 3: Intel Corporation.*
5. *IT Resource Center: Hewlett Packard.*
6. *Itanium Processor Family Error Handling Guide: Intel Corporation.*
7. *Johnson, T and McIntosh. Optimizing Itanium-based Applications. Paper published April 16, 2002.*
8. *Naffziger, S and Hammond. The Implementation of the Next Generation of 64b Itanium Microprocessor. Paper published for IEEE/ISSCC 2002, San Francisco, California.*
9. *Oracle Applications Standard Benchmark: www.oracle.com/apps_benchmark.*
10. *Oracle9i Administrator's Reference, Release 2 (9.2.0.1.0) for UNIX Systems: Oracle Corporation.*
11. *Oracle9i Concepts, Release 2 (9.2): Oracle Corporation.*
12. *Oracle9i Database Administrator's Guide, Release 2 (9.2): Oracle Corporation.*
13. *Oracle9i Data Guard Concepts and Administration, Release 2 (9.2): Oracle Corporation.*
14. *Oracle9i Data Warehousing Guide, Release 2 (9.2): Oracle Corporation.*
15. *Oracle9i Developer's Release Installation Guide, Release 9.0.1.0.1 for HP-UX 11i version 1.5 and Linux Intel on Itanium.*
16. *Oracle9i Release 2 (9.2.0.1.0) for HP-UX 11i version 1.6 on Itanium 2.*
17. *Oracle9i Installation Guide, Release 2 (9.2.0.1.0) for UNIX Systems: Oracle Corporation.*
18. *Oracle9i New Features, Release 2 (9.2): Oracle Corporation.*
19. *Oracle9i OLAP User's Guide, Release 2 (9.2): Oracle Corporation.*
20. *Oracle9i Performance Tuning Guide and Reference, Release 2 (9.2): Oracle Corporation.*
21. *Oracle9i Real Application Clusters Concepts, Release 2 (9.2): Oracle Corporation.*
22. *Oracle9i Real Application Clusters - Real Application Clusters Guard I - Concepts and Administration, Release 2 (9.2): Oracle Corporation.*
23. *Oracle9i Real Application Clusters High Availability Extension. (RAC Guard II). Software CD: Oracle Corporation.*
24. *Oracle9i Recovery Manager Reference, Release 2 (9.2): Oracle Corporation.*
25. *Oracle9i Release Notes, Release 2 (9.2.0.1.0) for HP Series 9000 HP-UX (64-bit).*
26. *Oracle9i Security Overview, Release 2 (9.2): Oracle Corporation.*
27. *Oracle9i Streams, Release 2 (9.2): Oracle Corporation.*
28. *Oracle9i XML Database Developer's Guide - Oracle XML DB Release 2 (9.2): Oracle Corporation.*
29. *Ramasamy, V and Hundt. Dynamic Binary Instrumentation for Intel® Itanium™ Processor Family. Paper published for ACM/MICRO34, December 2001, Austin, Texas.*
30. *Riedlinger, R and Grutkowski. The High Bandwidth, 256KB 2nd-level cache on an Itanium Microprocessor. Paper published for IEEE/ISSCC 2002, San Francisco, California.*
31. *Sarkar, S. Optimal Oracle9i on HP-UX 11i. Paper published for HP World Conference, 2001, Chicago, Illinois.*
32. *Transaction Processing Council: www.tpc.org.*
33. *Zahir, R, Ross and Morris. OS and Compiler Considerations in the Design of the IA-64 Architecture. Paper published for ACM/ASPLOS-IX, November 2000, Cambridge, Massachusetts.*