Native 64-bit Virtual Addressing for Oracle Rdb Row Caches Capability and Performance

Norman Lastovica
Oracle Corporation, Relational Technology Group, Rdb Engineering
Oracle New England Development Center, Nashua, New Hampshire

Oracle Rdb Release 7.1.2 Row Cache Enhancements

Oracle Rdb for OpenVMS Alpha Release 7.1.2 introduces two significant enhancements to the Row Cache feature: Snapshots in Row Cache and Native 64-bit addressing support for Row Cache. These features provide significant additional database performance and capability increases by further avoiding disk I/O and database page locking operations and by permitting significantly more data to be easily cached in memory.

Using the snapshots in Row Cache feature allows applications to approach zero disk I/O operations per transaction by reading and updating database rows entirely within memory while using the after-image journal to provide persistent storage data protection. Native 64-bit addressing support for Row Caches permits a vast number of database rows to be cached, limited solely by the amount of memory available in the computer.

This paper provides an introduction to the Native 64-bit addressing support for the Row Cache feature in Oracle Rdb Release 7.1.2. In addition, the results of large system performance and capability tests are described.

Row Cache Background

Introduced in Oracle Rdb Release 7.0, the Row Cache feature allows faster and more efficient access to database rows (both table data and index information). An Oracle Rdb Row Cache is a section of globally accessible memory that contains copies of database rows. This cache provides the ability to store, fetch, and modify frequently accessed rows in memory, avoiding disk I/O and locking. Database rows remain in memory even when the associated page has been removed from the local or global buffer pool. Row caching provides the following advantages:

- Reduced database read and write I/O operations
- Reduced database page locking operations
- Reduced CPU overhead for accessing a database row in cache
- Improved response time
- Efficient use of system memory resources for shared data

When a row is requested from the database, Oracle Rdb first checks to see if the requested table or index is mapped to an existing Row Cache. If a Row Cache is mapped, Oracle Rdb then checks to see if the requested row is in the cache. If the row is found in the Row Cache, the row is retrieved directly from the cache. If the row is not in the cache, Oracle Rdb checks the page buffer pool (either a local, per-process buffer pool or a global, shared buffer pool). If the row is not in the page buffer pool, Oracle Rdb locks the database page and initiates an OpenVMS I/O request

(either via $QIO or $IO_PERFORM) to retrieve the database page containing the row. Assuming space is available, the requested row is then inserted into the Row Cache.

When a modification is made to a cached row (either table data or index structures), if the original database page containing the row is locked for update in the process's page buffer pool (in such case, the page will have to be written back to the database anyhow), the modified data is written back to the database and to the cache. However, if the original database page is not locked for update or is not found in the process' buffer pool, the modification is made directly into the cache without incurring an additional database I/O or page locking.

Multiple Row Caches can be active for a database. A Row Cache must be available to, and shared by, all processes attached to the database. Using OpenVMS Galaxy cluster configurations, processes on different nodes within the galaxy share Row Caches that are stored in "galactic" shared memory.

No application changes are necessary when utilizing the Oracle Rdb Row Cache feature. The following are requirements for using the Row Cache feature:

- After-image journaling must be enabled for the database
  - Journals must be created
  - The fast commit feature must be enabled
- The Row Cache feature must be enabled
- Cache slots must be reserved
- Caches must be defined
- All database users must have access to shared memory. This can be accomplished by either:
  - Setting the database parameter NUMBER OF CLUSTER NODES to 1, or
  - Setting the database parameters GALAXY SUPPORT IS ENABLED and SINGLE INSTANCE

Existing VLM Capabilities in Oracle Rdb

The Oracle Rdb VLM (Very Large Memory) feature was created for Rdb Release 7.0 to allow access to more than 32 bits worth of virtual address space. This interface was implemented because, at that time, programs on OpenVMS Alpha did not have the ability to directly access memory beyond the 32-bit virtual address space.

The traditional VLM implementation within Oracle Rdb manipulated the OpenVMS Page Table Entry (PTE) mapping pointers for a small number of virtual pages in 32-bit process virtual address space (called "windows") to access physical pages allocated directly from OpenVMS. This technique allowed a large amount of physical memory to be accessed by changing memory pointers at run time.

While effective, this original VLM implementation did have some drawbacks. Performance was reduced, slightly, during the manipulation of the page table entries, as CPU cycles were required to unmap and remap a page of memory. And the intimate knowledge of OpenVMS internal memory management structures required additional maintenance efforts and was considered an issue for porting Oracle Rdb to the OpenVMS Itanium architecture.

Further, when using the VLM feature in prior versions of Oracle Rdb, Row Cache control structures were always stored in 32-bit virtual address space. The location of these control structures depended on whether shared memory for the Row Cache was designated as "process" or

"system." When using the SHARED MEMORY IS PROCESS attribute, the control structures were stored in a process global section in P0 (32-bit program region) virtual address space. When using the SHARED MEMORY IS SYSTEM feature, the control structures were stored in S0S1 (32-bit system region) virtual address space. This virtual memory in system (S0S1) virtual address space is allocated via the LDR_STD$ALLOC_PT system routine. Physical memory is allocated, one page at a time, by the MMG_STD$ALLOC_PFN_64 system routine.

Limitations Within a 32-bit Virtual Address Space

The Row Cache feature was historically limited to something less than 33 million total cached rows per database. This limitation was due to storing control data structures in 32-bit virtual address space. Even with the LARGE MEMORY IS ENABLED attribute, some data structures still were located in 32-bit virtual address space, ultimately leading to this restriction. As databases and application performance requirements have grown, this limitation has become a concern. In addition to Row Cache size limits, competition for the 1 GB process P0 virtual address space (shared among application code, application data buffers, database code, database buffers, Row Caches, and so on) and competition for the 2GB system S0S1 virtual address space (used by OpenVMS as well as for Row Caches and global buffers using the SHARED MEMORY IS SYSTEM feature) caused excessive compromise in design and performance.

Native 64-bit Virtual Addressing for Row Caches

OpenVMS introduced native support for accessing the 64-bit virtual address space defined by the Alpha architecture and has provided additional memory management features since 1995. The OpenVMS operating system and current Alpha architecture implementations support 8 TB (terabytes) of virtual address space.

Starting with Oracle Rdb Release 7.1.2, the Row Cache feature utilizes the native 64-bit virtual addressing support within the Alpha processor and OpenVMS operating system. Row Caches are always created in the OpenVMS P2 **(**64-bit program region) virtual address space.

Performance benefits of the native 64-bit addressing for Row Caches are realized by allowing vastly larger Row Caches to be created and by avoiding performance penalties related to the previously available Row Cache use of VLM capabilities within Oracle Rdb. Additionally, configuration and management of very large Row Caches (those with many or large cache slots) is simplified by always utilizing OpenVMS global sections.

There are no visible user or application effects (beyond potential performance improvements) due to the Oracle Rdb implementation of native 64-bit virtual addressing for Row Caches.

The Row Cache memory mapping attributes LARGE MEMORY IS ENABLED and SHARED MEMORY IS SYSTEM have been replaced with the RESIDENT attribute. Although these clauses are now deprecated, these attributes are internally considered as synonyms for RESIDENT.

If the RESIDENT attribute is specified (implicitly or explicitly) for a Row Cache, the cache will be created as a memory-resident global section. If the section is at least as large as the number of CPU-specific pages mapped by a single page table page (approximately 8mb), it will be created utilizing OpenVMS "shared page tables" (potentially with granularity hints).

OpenVMS *shared page tables* for memory-resident global sections reduce physical memory consumption by allowing multiple processes to share one set of page table entries for the global section. These shared page tables can save a significant amount of physical memory for large global sections with many users. The Alpha processor's implementation of *granularity hints* allows

ranges of pages to be mapped by a single translation buffer entry within the processor, leading to improved translation buffer hit rates and utilization.

Considering the reduction in physical memory consumption provided by *shared page tables,* and the performance advantages provided *granularity hints*, Oracle recommends that Row Caches be configured with the RESIDENT attribute when possible. It is important, however, to avoid "starving" the system for memory by creating resident caches that consume memory that would be needed by, for example, application program working sets.

Global sections are created with the $CRMPSC_GDZRO_64 system service by the first process to reference a Row Cache. Subsequent access to a created cache by other processes is with the $MGBLSC_64 system service. Row Cache global sections are always mapped in P2 virtual address space (either in the default P2 region or a specified region if shared page tables are being created). Global sections are named including a system-unique database specification along with a unique Row Cache identifier within the database.

When a memory resident global section is created, OpenVMS writes invalid global PTEs (Page Table Entries) to the global page table. When the global section is mapped, invalid page table entries are placed in the page table; physical memory is not allocated until the pages are referenced. To ensure that all pages are valid and resident in memory, Oracle Rdb accesses each page when a memory resident section is first created. The first access to the page causes a page fault (due to the invalid PTE) and OpenVMS then initializes the physical page with binary zeros and the page is then valid for all users of the global section.

Specify Physical Memory by RAD

On AlphaServer GS-series systems that support OpenVMS Resource Affinity Domains (RADs), Row Caches that are configured with the RESIDENT attribute can be individually designated with a preferred RAD for memory allocation. By carefully managing the home RADs for database users with critically high performance requirements and the physical memory allocated for Row Caches used by these processes to appropriate RADs, performance can potentially be increased by reducing "off-RAD" memory references. Note that the RAD attribute for a Row Cache simply specifies a hint to OpenVMS about where memory should be physically allocated. It is possible that, if the memory is not available, it will be allocated from other RADs in the system.

Restrictions

This change to utilize native 64-bit virtual addressing is specific to the Row Cache feature in this release of Oracle Rdb.

With Oracle Rdb Release 7.1, the maximum total size of any individual Row Cache is 2,147,483,647 rows. This restriction may be relaxed in a future Oracle Rdb release. A database may be configured with many Row Caches so the total number of cached rows for a database is generally limited by the physical memory available on the system.

System Parameter Considerations

Shared memory sections using the LARGE MEMORY IS ENABLED or SHARED MEMORY IS SYSTEM features were previously not created as OpenVMS global sections and were not directly affected by the various system parameters controlling global section allocation. Because all Row Cache shared memory sections are now created and mapped as global sections, it is possible that the GBLSECTIONS, GBLPAGES and GBLPAGFIL system parameters may have to be increased in order to map large caches that previously relied on the LARGE MEMORY IS ENABLED or SHARED

MEMORY IS SYSTEM features. Note that GBLPAGES and GBLPAGFIL are dynamic system parameters (can be updated without rebooting the system) while the updates to the GBLSECTIONS parameter requires that the system be rebooted for the change to take effect.

When mapping Row Caches or opening databases using the GALAXY SUPPORT IS ENABLED feature in an OpenVMS Galaxy environment, a "SYSTEM-F-INSFMEM, insufficient dynamic memory" error may be received. One source of this situation is running out of OpenVMS Galaxy "Shared Memory regions" (a future OpenVMS release is expected to provide a more specific error message). For OpenVMS Galaxy configurations, the system parameter GLX_SHM_REG specifies the number of "shared memory region" structures configured into the Galaxy Management Database (GMDB). While the default value of 64 (for OpenVMS versions through at least V7.3-2) may be adequate for some installations, sites using a larger number of databases or Row Caches will likely find the default insufficient. Refer to the Oracle Rdb Release Notes for additional information about configuring an OpenVMS Galaxy environment for Oracle Rdb.

The RMU /DUMP /HEADER command can be used to display the physical global section sizes for Row Caches configured in the database. Oracle also offers a Row Cache sizing and configuration tool available for download from Oracle Support's MetaLink.

A Test Using Very Large Caches

To help test and demonstrate the capabilities of a rather large amount of cached data, a test database with one billion (10^9; 1,000,000,000) rows and associated indexes in Row Caches was created. This database was configured with 10 tables each horizontally partitioned across 5 storage areas. Each table consists of 10 quadword (64-bit integer) columns. One pseudo-ranked sorted index on a single column was created for each table. 100,000,000 data rows of random content were then loaded into each table. Each table required approximately 1.2 million index nodes at 2000 bytes each (index depth of 5 levels).

For each table, two caches were configured: one for the rows and one for the index nodes. Simulating an environment planning for future database growth, caches for the database rows were configured with 101,000,000 "slots" per table and the caches for the index nodes were configured with 1,515,000 "slots" per index. The caches for the indexes were somewhat over-sized as compared to the caches for the data rows. This was intentional to simulate a typical production application where the DBA would allow for modifying index key values. Changes to the index keys could cause index node splitting, leading to additional index nodes. The caches for indexes were also configured specifying the Row Cache "no replacement" attribute to enable an optimization that helps avoid cache slot contention for heavily accessed database objects.

Taking advantage of the "Snapshots in Row Cache" feature of Oracle Rdb Release 7.1.2, each cache was created with 50,000 snapshot "slots" to allow read-write transactions to store snapshot records for modified rows directly into cache (avoiding disk I/O and database page locking). By eliminating snapshot and database I/O when modifying cached records, the application should more fully utilize the system CPU resources, leading to reduced transaction duration. All caches were created specifying the "SHARED MEMORY IS PROCESS RESIDENT" attribute to allow use of OpenVMS shared page tables and granularity hint regions (reducing physical memory consumption and improving performance).

The total amount of physical memory utilized for Row Caches for this configuration was approximately 202GB (representing both overhead and usable data storage).

Data content for each table was generated by a program that wrote records of random values into an OpenVMS mailbox; the mailbox was read by an RMU /LOAD session that populated the table.

After populating the caches with the data row and index node content, a synthetic workload was run. This workload represented database server processes that would continuously execute database update transactions, with each transaction choosing a table at random and selecting and modifying a random row in the table. Due to the initial data distribution pattern, approximately one transaction in a thousand would find the specified row non-existent and would insert the row into the database. Thus, every transaction either updates an existing database row or, occasionally, inserts a new row into the database.

The workload program was written in C using the Oracle Rdb SQL precompiler. In order to keep the test program development effort limited, and to help ensure a reasonable percentage of non-database activity on the system, the SQL update and insert statements are generated at run-time using the dynamic SQL interface. The technique of using entirely dynamic SQL for the workload may be revisited in the future, as the CPU cost to completely interpret and compile the SQL statement for each operation may be an excessive burden on performance of this test and limited overall throughput. The Oracle Rdb SQL language provides the ability to, at run time, compile and later repetitively execute statements. This approach would be expected to save some CPU time.

To ensure high levels of performance for writing to the after-image journal (AIJ), the database was configured to use the optional AIJ Log Server (ALS) process. The ALS process removes the task of writing to the AIJ from the user processes by performing all AIJ write operations for a database. The ALS is optimized to avoid journal locking operations, and does "double buffered" asynchronous I/O operations to the after-image journal to reduce effective I/O latency.

As the test application would be modifying or inserting records for every transaction, a large number of locking operations were expected during the test. To provide for efficient system scaling, OpenVMS was configured to use the optional dedicated CPU lock manager. This feature provides significant performance increases for heavy lock operation loads when many CPUs are present in the system by eliminating contention for locking data structures. A single CPU is dedicated to locking operations for all processes on the system with very low overhead.

System Configuration

The test system utilized a Hewlett-Packard AlphaServer GS1280 running 32 Alpha EV7 processors at 1150 MHz with 256GB of physical memory. OpenVMS Alpha Version 7.3-2 base level X9Z0 was installed and disk volumes were configured as RAID-5 arrays in an EVA12000 storage server accessed through KGPSA controllers. As disk I/O loads were expected to remain quite low during this test (ultimately less than 5000 I/O write operations per second), the configuration and absolute performance of the storage subsystem was not a significant concern and little analysis or tuning was performed or required.

Initial Test Results – More than 13,000 Database Transactions Per Second

To simulate a large multi-client database server environment, multiple copies of the workload were run at the same time. Instances of the program were added (to a total of 60) until the system was effectively CPU bound (over 98% utilization). All database server workload instances solely executed read-write transactions and updated or inserted a single row per transaction.

At this workload level, the peak database transaction rate was 13,228 transactions per second with significantly less than 1 disk I/O request per transaction. System-wide disk I/O rate was approximately 3000 per second and the system-wide locking operation rate (new lock requests, promotions, demotions and releases performed by the OpenVMS lock manager) was approximately 150,000 per second.

Second Test

In an attempt to increase the transaction rate, a second experiment was run with database snapshots disabled and the Oracle Rdb "commit to journal" feature enabled. Avoiding writing snapshots to the Row Caches should reduce CPU consumption (in a typical production system, this would not often be a viable configuration because it causes read-only transactions to get promoted to read-write transactions and some operations such as online database backup are prohibited while snapshots are disabled). The "commit to journal" feature reduces database root-file I/O by allowing transactions to allocate transaction sequence numbers from the database in groups of up to 1024 at a time, rather than one per transaction.

The database was also configured to allow Oracle Rdb to use the OpenVMS "Fast I/O" feature. Setting the "buffer object enabled" attribute for database file, root file, after-image journal file, and recovery-unit journal file I/O operations enables this feature within Oracle Rdb. The "Fast I/O" feature provides a streamlined I/O interface that can reduce CPU usage by simplifying the I/O path and performing buffer locking and probing once (typically at application startup), rather than for each I/O request.

To enable multiple statistics global sections to be automatically created by Oracle Rdb, OpenVMS was also configured to enable RAD support on the GS1280. By using many statistics global sections, memory contention for statistics counters maintained by Oracle Rdb is reduced. When Oracle Rdb detects that a system is configured with RAD support when a database is opened, a statistics section is created in each RAD. As user processes attach to the database, they select a statistics global section to be used based on the process's "home" RAD.

Second Test Results – One Million Database Transactions Per Minute

With the reduced overhead due to the elimination of database snapshots, the "commit to journal" feature, and the "FAST I/O" feature, the same test workload peaked approximately 17,000 database transactions per second. This was considered a significant data point in that it represents just over 1 million database transactions per minute for this particular workload.

Results Analysis and Updates

The performance indicators in this test caused some amount of interest between both HP and Oracle engineering staffs. Further lab testing revealed some areas in the Rdb engine that could be further optimized. In particular, some alignment faults (cases where the instruction stream was expecting to operate on naturally aligned data cells but caused a fault when the data was not aligned) were eliminated from the code.

The application itself was also enhanced to pre-compile all possible update statements during program startup. Then at run time, the correct pre-compiled statement was executed during each transaction. This technique avoided having to compile the dynamic SQL statement for each update, saving CPU cycles.

Third Test

Each release of Oracle Rdb version 7.1 is currently shipping as two variants: one compiled for all Alpha processors and one compiled for Alpha EV56 and later processors. The EV56 variant includes code compiled to utilize the Alpha byte-word instructions. As an internal performance experiment, the Rdb code was compiled explicitly for the EV67 and later Alpha processors. This configuration allowed the language compilers to produce an Alpha instruction sequence that was

optimal for the EV67/EV68/EV7 processors in both use of available instructions and the scheduling of instructions for the quad-issue Alpha processor.

A 32 processor GS1280 with 128GB was configured and made available for a second week of testing in January of 2004. OpenVMS V7.3-2 was installed with the experimental compilation of Oracle Rdb along with the enhanced workload program. The OpenVMS feature RAD support was not enabled for this test.

A number of experiments were run with the updated configuration. Areas of interest included measuring performance with varied numbers of CPUs to determine how effectively the system scaled from 8 to 32 processors, the effect of the Rdb AIJ Log Server (RDMALS) process, impact of the Record Cache Server (RDMRCS) process checkpoint operations, and so on.

Third Test Results – 1,791,480 Database Transactions Per Minute
A sustained measured database transaction rate of 1,791,480 for an interval greater than one minute was demonstrated (representing 29,858 transactions per second). Rates of over 30,000 transactions per second were sustained over 15 second intervals.

Further Analysis

In addition to the performance indicators, the ability to run these workloads on a large multi-processor system also provided the opportunity for Oracle Rdb Engineering and Hewlett-Packard OpenVMS engineering to measure and review various performance indicators. The results of such analysis helps provide direction for further areas of investigation for performance enhancements for future releases of both OpenVMS and Oracle Rdb.

Observations and Recommendations

Using caches as large as those in the tests requires some operational considerations. First, it can take quite a while for the Row Cache Server (RCS) process to create and map (and, by extension, initialize) these large global sections. The Database Recovery process (DBR), as well, can run for a longer time because it must scan all cache slots in order to make sure that the failed process had not reserved any cache slots. It may be possible to reduce these effects, to some extent, in future releases of Oracle Rdb.

In the same way, the RMU/SHOW STATISTICS utility can take a long time to accumulate statistics information for very large caches. Some of the displays require that all cache slots be examined (to determine the amount of space used in a cache, or to count the number of reserved rows, for example). This can require a significant amount of CPU time and the actual screen refresh interval will be quite slow.

When closing, backing up, or analyzing a database, the RCS process must write all modified rows back to the physical database. When using huge caches with many modified rows in cache, the RCS may take quite a while to do this writing. Planning ahead for a shutdown or backup may allow you to perform an RMU /SERVER RECORD_CACHE CHECKPOINT command prior to the scheduled shutdown or backup time. This allows the RCS to get a “head start” writing modified rows back to the database.

Additional Information

For more information about Oracle Rdb performance and configuration options, you may wish to refer to the following documents available from Oracle Corporation:

- *Oracle Rdb Release 7.1.2 Release Notes*

- *Oracle Rdb7 Guide to Database Performance and Tuning*
- *Oracle Rdb7 Guide to Database Design and Definition*

For detailed information about the Alpha processor or OpenVMS memory management and programming, you may wish to refer to the following documents available from Hewlett-Packard:

- *OpenVMS Programming Concepts Manual*
- *HP OpenVMS System Services Reference Manual*
- *OpenVMS Calling Standard*
- *HP OpenVMS System Manager's Manual*
- *HP OpenVMS Alpha Partitioning and Galaxy Guide*
- *Alpha Architecture Handbook*
- *Alpha Microprocessor Hardware Reference Manuals*

Acknowledgements

Running the performance experiments would not have been possible without the expert assistance of Craig Showers and Bill Gabor at Hewlett-Packard's OpenVMS Enterprise Solution Center in Nashua, New Hampshire for providing access to the system and storage configuration along with system and operational support. Special thanks as well are due to Jeff Jalbert and Tom Musson of JCC Consulting, Bill Gettys, Bill Wright, Ian Smith, Ed Fisher, and Paul Mead of Oracle Rdb Engineering, and to Christian Moser, Greg Jordan, Karen Noel, Sue Lewis, Kevin Jenkins, Steve Lieman, and Tom Cafarella of Hewlett-Packard OpenVMS engineering for providing outstanding technical support and enthusiasm.