Guide to Database Performance and Tuning: Row Cache Enhancements

A feature of Oracle Rdb

By Norman Lastovica
Oracle Rdb Relational Technology Group
Oracle Corporation

The Rdb Technical Corner is a regular feature of the Oracle Rdb Web Journal. The examples in this article use SQL language from Oracle Rdb V7.1.2 and later versions.

Guide to Database Performance and Tuning: Row Cache Enhancements

Oracle Rdb 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 can be combined to provide significant additional database performance and capability increases by further avoiding disk I/O and 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 only by the amount of memory available in the computer.

This paper provides an introduction to these two enhancements.

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). The Oracle Rdb Row Cache is a section of globally accessible memory that contains copies of database rows providing 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 flushed back to disk. Row caching provides the following advantages:

- Reduced database read and write I/O operations
- Reduced database page locking operations
- Improved response time
- Efficient use of system resources (memory) 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. If the row is not in the page buffer pool, Oracle Rdb performs disk I/O to retrieve the row. 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' 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 database I/O or page locking.

Multiple row caches can be active for a database. A row cache is available to, and shared by all processes attached to the database. On OpenVMS Galaxy configurations, processes on different nodes 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 is** 1
  - Setting the database parameter **galaxy support is enabled**

Snapshots in Row Cache

The snapshot mechanism within Oracle Rdb allows read-only transactions to see a consistent, non-changing view of the database while other transactions update the database. The prior versions of database rows are written to special snapshot areas of the database by transactions that update the rows and are then accessible to read-only transactions. These *snapshot* copies of rows can now be stored in shared memory rather than in the on-disk snapshot storage areas. Applications are able to more easily access and modify rows with no database page related operations. Read-only transactions can quickly access snapshot copies of rows from memory and read-write transactions can quickly write snapshots. The reading and writing of the snapshot information can be accomplished with no database page I/O or associated database page locking.

With the snapshots in row cache feature, read-write transactions are able to avoid database I/O and locking operations that are required to maintain snapshot pages and the pointers to the snapshot pages on the "live" database pages. By eliminating contention and I/O stalls, performance can be dramatically increased. Applications which previously required many page I/O operations when modifying rows can now run nearly "I/O-free" and more fully utilize the full speed of the system processor.

Snapshot copies of records are stored (in the database snapshot storage areas or in the snapshot portion of a row cache) in a "chain" such that the newest snapshots are stored at the head of the chain. A read-only transaction searches the chain for a database row for the first *visible*[1] occurrence of the row; there may be a number of copies of a particular database record stored in a snapshot chain.

Snapshot copies of a database record must be maintained until there are no transactions in the database older than the transaction that stored the snapshots of the record. As the oldest transactions in the database commit, space used to store snapshot records (either in the snapshot storage areas or in the snapshot portion of a cache) may be re-used for storing newer snapshots. By keeping transactions relatively short, the number of snapshot rows that need to be stored can be reduced.

Each row cache defined in a database can be designated to allow a specified number of snapshot rows to be stored in the cache. The number of snapshot rows allowed is specified in addition to the

[1] A read-only transaction can only return rows that were committed prior to the time that it started. The snapshot area might contain multiple updates for the same row and a check must be made to see which are *visible* to the current read-only transaction.

number of *live* database rows that can be stored in the cache. Since many versions of a row may be stored in the snapshot portion of the cache, the number of snapshot rows and *live* cache rows are specified independently. As the snapshot portion of the row cache is effectively an extension of the row cache it self, other attributes of the cache are shared with the snapshot portion.

The snapshot portion of a row cache may be larger (may contain more rows) or smaller (may contain fewer rows) than the live portion of the row cache. Because application and workload behavior determines the number of database rows that are modified and the relative transaction length, it is not feasible to make specific recommendations for sizing snapshot portions of caches for all application and database types. However, it is expected that the ratio of the size of the snapshot cache to the main cache may be similar to the ratio of the database snapshot storage area to the live storage area. If an application has long running transactions and active read-write transactions modifying cached data, many snapshot copies of the modified data may need to be maintained. Such caches may need many snapshot slots.

When the snapshot portion of a cache fills and no slots are available for re-use (due to the age of the oldest transaction in the database), read-write transactions may need to *overflow* snapshot records from cache to disk. This overflow operation can be quite costly in terms of CPU time and disk I/O operations. When a snapshot cache is discovered to be full and a read-write transaction must store a new snapshot copy of a row, all existing snapshot copies for that row must be written from the cache to the snapshot storage area on disk. Additionally, all future snapshot operations to the cache must also be written to disk.

When the snapshot portion of a row cache is marked as being *full*, the Row Cache Server (RCS) process periodically checks the cache to see if space is available for reuse. Similar to the algorithms governing space reclamation in snapshot storage areas, this space only becomes available when the oldest transaction in the database has completed. When the RCS process finds reclaimable space in the snapshot portion of a cache that is marked *full*, it will clear the *full* indicator to permit new snapshot copies to be stored in the cache by read-write transactions.

Even though one or more snapshot chains have been written from the cache to disk, the snapshot records remain in the cache and cannot be reclaimed until they are no longer needed by any transaction in the database.

When a row is removed from the cache (due to it becoming fragmented, growing too large for the cache, or from a **truncate table** operation[2]), all snapshot copies for the row must be written from cache back to the snapshot storage area on disk. This can be a relatively costly and slow operation

[2] While this paper has a focus on table rows, this feature applies also to index nodes, hash buckets, etc.

and can often be avoided by insuring that caches are sized with slots that are large enough for the data being stored.

At certain times during normal database operations, all modified rows must be written from cache[3] back to the physical database storage areas. Events that require writing all modified data back to the database include closing the database, backing up the database, an explicit checkpoint request and verifying the database.

When a database is configured with snapshots enabled and the **Snapshots in Row Cache** feature is not enabled, it is likely that very few modified rows remain in cache memory. This is due to updating the live database pages with snapshot pointers after the snapshots have been written to disk. Because the live database pages tend to still be in memory with an update lock when modified rows are written to the cache, the modifications are also written to the database pages. As these database pages already must be written to the database (with an updated snapshot pointer), Rdb puts the modified row content on the page as well, to avoid having the RCS write it later.

However, when caches are configured with snapshots in cache, the *live* cache itself may have many modified rows. When there are many modified rows in memory, it may take a significant amount of time to write all of these rows back to the database. You should allow sufficient time for modified rows to be written when planning for operations such as database backup and verification

Closing a database may now be a more time consuming operation. Oracle strongly recommends that databases using the Row Cache feature be specified with the **open is manual** attribute in order to allow the database administrator to control when open and close operations are executed.

Backing store (.RDC) files are used to provide persistent storage of modified rows from a cache. The backing store files are written during RCS “checkpoint” operations and are read during node-failure recovery (database being first opened after a system failure). When using the **Snapshots in Row Cache** feature, caches on database rows that are heavily modified should generally be set to checkpoint to the backing store files (this is the default configuration). It is generally faster and more efficient to checkpoint modified rows to the backing store files than to the database storage areas.

Applications may tend to experience improved performance by removing delays introduced by disk I/O operations. Some systems, therefore, may see a significant reduction in system idle time due to the reduction in I/O stalls. Presumably, this will be reflected in an increase of over-all application performance and throughput as the computer system is now being more effectively utilized.

[3] There may be many actives row caches in the database.

Several SQL database statements have been enhanced to allow the new snapshots in row cache feature to be enabled. The **create cache** clause of the IMPORT and CREATE DATABASE statements, as well as the **add cache** and **alter cache** clauses of the ALTER DATABASE statement can be used to set parameters for the snapshot portion of a row cache. The SQL syntax to support snapshots in row cache is:

```
ROW SNAPSHOT IS { DISABLED | { ENABLED [ ( CACHE SIZE IS n ROWS) ] } }
```

The **row snapshot is disabled** clause disables storing snapshot copies of rows within the cache. The **row snapshot is enabled (cache size is** n **rows)** clause enables storage of snapshot copies of rows within the cache and specifies the number of snapshot *slots* to allocate for the cache. The default for all caches is to have the snapshots in row cache feature disabled.

The following example demonstrates using SQL to modify the cache named "SALES" to disable storage of snapshot rows in cache and to enable the cache named "CLEARING" for storage of up to 12,345 snapshot rows in the cache:

```
SQL> ALTER DATABASE FILE MYDB
cont>   ALTER CACHE SALES
cont>      ROW SNAPSHOT IS DISABLED;
SQL> ALTER DATABASE FILE MYDB
cont>   ALTER CACHE CLEARING
cont>      ROW SNAPSHOT IS ENABLED (CACHE SIZE IS 12345 ROWS);
```

The RMU /SET ROW_CACHE command has been enhanced as well to allow the new snapshots in row cache feature settings to be manipulated. The “/ALTER=(...)” qualifier now accepts a “SNAPSHOT_SLOT_COUNT=n” keyword. The value “n” specified sets the number of snapshot slots in the cache. A value of zero disables the snapshots in row cache feature for the specified cache.

The following example modifies the database OMTXR to set the snapshot slot count for the cache “EMPL_IDX” to 25000 slots and disables snapshots in cache for the “WEEKLY” cache:

```
$ RMU /SET ROW_CACHE OMTXR –
 /ALTER=(NAME=EMPL_IDX, SNAPSHOT_COUNT=25000) –
 /ALTER=(NAME=WEEKLY, SNAPSHOT_COUNT=0)
```

In order to use the Row Cache feature, you must enable the Oracle Rdb Fast Commit feature. When the **Fast Commit** feature is utilized, only the after-image journal (AIJ) is guaranteed to contain saved database changes when a transaction commits. Protecting the after-image journal files is thus very important. Oracle encourages use of data protection features such as disk volume shadowing and especially the Oracle Rdb Hot Standby feature to help ensure the safety of the after-image journal content.

The Row Cache *backing store* (also known as .RDC) files are also important to ensure rapid database recovery after a system failure. These files contain the modified row content of caches from the most recent row cache server (RCS) checkpoint operation. Oracle encourages use of data protection features such as disk volume shadowing to help ensure the safety of the contents of the *backing store* files.

The **Snapshots in Row Cache** feature tends to benefit row (index nodes or user data records) modifications more than inserts. Inserting a row in the database requires that space be allocated within a database storage area for the row itself and to allocate a unique database key (DBKEY). As the "live" database storage area is modified to allocate a new database row, the snapshot storage area is updated as well (to indicate that the row previously did not exist in the database). Cached sorted indexes on tables where rows are being inserted benefit from the snapshots in row cache feature.

The **RMU /DUMP /HEADER=ROW_CACHE** command can be used to display the configuration for the row caches defined for a database. The following example shows the snapshot slot count and the "live" slot count for the cache called TICKET_IDX_MASTER:

```
Row cache "TICKET_IDX_MASTER"
    Cache ID number is 6
    Allocation...
      - Row slot count is 100000
        - Snapshot slot count is 15000
        - Snapshots in cache enabled
      - Maximum row size allowed in cache is 960 bytes
```

The **RMU /SHOW STATISTICS** utility has been enhanced to provide for real-time monitoring of the snapshots in row cache feature, including displays containing the utilization of the snapshot portion within a row cache. The "Row Cache Status" screen displays the number of snapshots slots for a cache, the number that are in use, and the number that can be reclaimed (a snapshot slot containing a row can be reclaimed if the snapshot was written by a transaction older than the oldest active transaction in the database). The "Row Cache" screen displays counters for the number of rows stored and retrieved from the snapshot portion of a row cache.

The following screen display from the RMU /SHOW STATISTICS utility demonstrates the additional information on the "Row Cache Status" screen. In this example, the cache named "C5" has been configured with 500,000 "live" cache slots and 10,000 snapshot slots. 9,742 of the slots contain snapshot data and 9,514 of the snapshot slots can be "reclaimed" (reused to store other snapshot rows because no transaction in the database could possibly need to use the existing snapshot copy). The "Snap Cursor" value indicates the starting position for searching for empty or reclaimable slots within the snapshot portion of the cache.

```
Node: CACHME (1/1/1)   Oracle Rdb V7.1-201 Perf. Monitor 7-AUG-2003 22:18:16.86
Rate: 3.00 Seconds             Row Cache Status           Elapsed: 00:23:04.79
Page: 1 of 1                DGA100:[BIGDB]BIGDB.RDB                 Mode:Online
-------------------------------------------------------------------------------
                                 For Cache: C5
Statistic.Name Stat.Value Percent                                    MRES

Total slots:      500000  100.0% Slot Length: 256  Hash slots: 524288
Slots full:          913    0.1% Use:        110  12.0%
Slots empty:      499087   99.8% Rsv:        118   0.0%
Marked Slots:        639    0.1% Hot:        639 100.0% Cold:          0   0.0%
Clean Slots:      499361   99.8% Hot:          0   0.0% Cold:     499361 100.0%
Used Space:          228k   0.1% Wstd:          5k  0.0%
Free Space:       127766k  99.8%
Hash Que Lengths:  Empty:523375 1:913      2:0        3:0        4+:0
Cursor position:   1092 of 500000 wrapped 0 times
Cache latched:     No
Cache is full:     No            Cache modified:    Yes  Snapshot is full: No
Checkpoints: 5 Last:  7-JUL-2003 15:35:25.71 Location: 142:39
Cache Recovery:    142:45212
Snap Slots:        10000  100.0% Ful:       9742  97.4% Rcl:       9514  95.1%
Snap Cursor:  5477 of 10000 (slot 505477) wrapped 1 time
```

For Oracle Rdb Release 7.1.2, snapshots are only maintained in row caches for those objects stored in uniform format storage areas. This restriction is due to complexities involved during sequential scans of mixed format storage areas and is expected to be relaxed in a future release of Oracle Rdb.

Native 64-bit Virtual Addressing for Row Caches

The Oracle Rdb VLM (Very Large Memory) feature was created in 1995 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, and caches of more than 1 GB (gigabyte) were required. Since then, HP OpenVMS has introduced native support for accessing the 64-bit virtual address space defined by the Alpha architecture and has provided additional memory management features. The OpenVMS operating system and current Alpha architecture implementations support 8 TB (terabytes) of virtual address space.

Within Oracle Rdb, the Row Cache feature was historically limited to something less than 33 million total cached rows per database. This limitation was due to a requirement to store some row cache control data structures in 32-bit virtual address space. Even when the LARGE MEMORY IS ENABLED attribute is set, some data structures had to be located in 32-bit virtual address space, leading to this restriction. As databases and application performance requirements have grown, this limitation has become of concern for some customers. In addition to row cache size limits, competition for the 1 GB process P0 virtual address space (shared between application code, application data buffers, database code, database buffers and row caches) caused excessive compromise in design and performance.

The traditional VLM implementation within Oracle Rdb manipulated PTE (Page Table Entry) 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. The technique allowed a large amount of physical memory to be accessed by changing memory pointers. While effective and widely utilized, the original VLM implementation used in Oracle Rdb had some drawbacks. Performance is reduced, slightly, during the manipulation of the page table entries, as a number of CPU cycles are required to unmap and remap a page of memory. Further, intimate knowledge of OpenVMS internal memory management structures requires additional maintenance efforts and was known to be an issue for porting Oracle Rdb to the OpenVMS Itanium architecture.

When using the **large memory is enabled** 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 is **process** or **system**. With **shared memory is process**, the control structures were stored in a process global section. With **shared memory is system**, the control structures were stored in system 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 now always created in the OpenVMS P2 **(**64-bit program region - 64-bit addressable process space) 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 existing Very Large Memory capabilities within Oracle Rdb. Configuration and management of very large row caches (those with many or large cache slots) is simplified by always utilizing global sections.

There should be no visible user or application effects due to the Oracle Rdb implementation of native 64-bit virtual addressing for Row Caches. If the RESIDENT attribute is specified for a row cache, the cache will be created as a memory-resident global section 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 page table entries for the global section. 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.

The row cache memory mapping features **large memory is enabled** and **shared memory is system** have been replaced with the RESIDENT attribute. If the **large memory is enabled** or **shared memory is system** attributes are specified for a row cache, the cache is considered to be RESIDENT. Although these clauses are now deprecated, **large memory is enabled** and **shared memory is system** are internally considered as synonyms for "RESIDENT".

This change to utilize native 64-bit virtual addressing is specific to the Row Cache feature in this release of Oracle Rdb. Rdb database global buffers, for example, continue to be mapped into 32-bit virtual address space and continue to support the **large memory is enabled** and **shared memory is system** attributes. No additional support for 64-bit virtual addressing (including via callable interfaces such as SQL) is provided in this release of Oracle Rdb. Oracle is considering expanding uses of native 64-bit virtual addressing within Oracle Rdb for future releases.

With Oracle Rdb Release 7.1.2, the maximum total size of any individual row cache (the combination of *live* rows and snapshot rows) 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.

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 global section system parameters. 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 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 process 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 created.

When mapping record caches or opening databases using the **galaxy support is enabled** feature, a “SYSTEM-F-INSFMEM, insufficient dynamic memory” error may be encountered. One source of this problem is running out of Galaxy Shared Memory regions. For Galaxy systems, GLX_SHM_REG is the number of shared memory region structures configured into the Galaxy Management Database (GMDB). While the default value (for OpenVMS versions through at least V7.3-1) of 64 might be adequate for some installations, sites using a larger number of databases or row caches may find the default insufficient.

Oracle recommends that if a “SYSTEM-F-INSFMEM, insufficient dynamic memory” fatal error is returned when mapping record caches or opening databases in an OpenVMS galaxy environment, the GLX_SHM_REG parameter be increased. A recommended starting value for the GLX_SHM_REG parameter is at least 2 times the sum of the number of Galaxy-shared row caches and number of databases that might be accessed at one time. As the Galaxy shared memory “region” structures are not very large, setting this parameter to a higher than required value does not consume a significant amount of physical memory. This parameter must be set on all nodes in the Galaxy. Changing the GLX_SHM_REG system parameter requires that all nodes in the Galaxy be shut down and the Galaxy reformed by booting each instance. Refer to the OpenVMS documentation for additional information.

The **RMU /DUMP /HEADER=ROW_CACHE** command can be used to display the configuration for the row caches defined for a database. The following example shows the slot counts and slot length as well as the amount of memory that the memory-resident global section will consume for the cache named “SERVICE_REQUEST”:

```
Row cache "SERVICE_REQUEST"
    Cache ID number is 6
    Allocation...
      - Row slot count is 100000
        - Snapshot slot count is 15000
        - Snapshots in cache enabled
      - Maximum row size allowed in cache is 960 bytes
      .
      .
      .
    Shared Memory...
      - Shared memory will be mapped resident (OpenVMS Alpha only)
      - Shared memory section requirement is 116,465,664 bytes
```

For additional information about the Alpha processor or OpenVMS memory management and programming, you may wish to refer to the following documents available from HP:

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

Oracle Rdb
Guide to Database Performance and Tuning: Row Cache Enhancements
August 2003

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com