Some Practical Considerations When Using Rdb “Row Caches”

Jeffrey S. Jalbert,
Thomas H. Musson,
Keith W. Hare

JCC Consulting, Inc.

Abstract

The introduction of "Row Caches" in Rdb 7.0 gives the physical database designer another and powerful tool to manage the performance of an Rdb database. Row caches interact strongly with VMS and therefore the limitations of the VMS operating system will strongly affect the way caches can be designed and deployed.

This presentation will address a number of practical issues relating to row cache design, how to configure VMS for that design and how to select what database objects should be cached.

What is a Row Cache?

- A  bullet for the physical database designer
  - Can use extremely large amounts physical of memory
- Running on the back of VMS which limits that
  - 1970's 32-bit operating system, one or two Gb addressing limits
  - Limiting with current machines supporting 5, 10, 16, 32 GB of memory and more

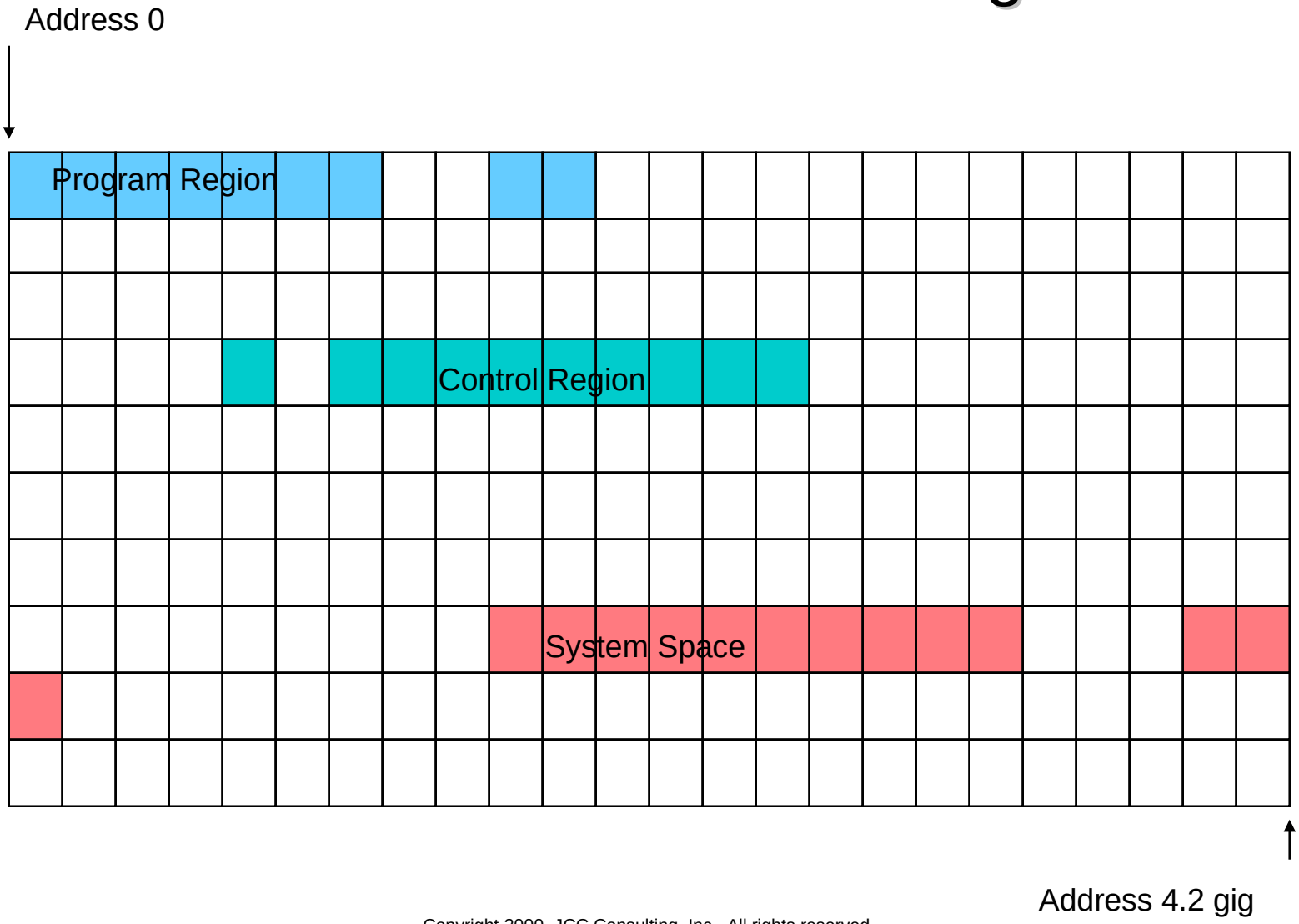
What is a Row Cache?

- Basically this is another way to store data in memory instead of having to read it from disk
- Very different paradigm from the traditional Rdb buffer mentality
  - But compatible with that
- Database buffers and pages are still present as an ultimate conceptual platform.

VMS Addressing

- It is important to understand a couple of things about VMS and memory management while discussion row caches
- Each 32-bit address is partitioned into 3 sections
- Bit 31 determines whether the address is “system” or process.
- Bit 30 partitions process address space into program and control regions
  - Control region is where DCL and process stacks and other such stuff live
    - Could contain “other stuff” as desired
  - Program region is where code usually resides
- Every process maps the same pages into system space
- On Alpha systems with VMS 7.0, two additional memory sections are added, P2 and S2. These are 64-bit addresses

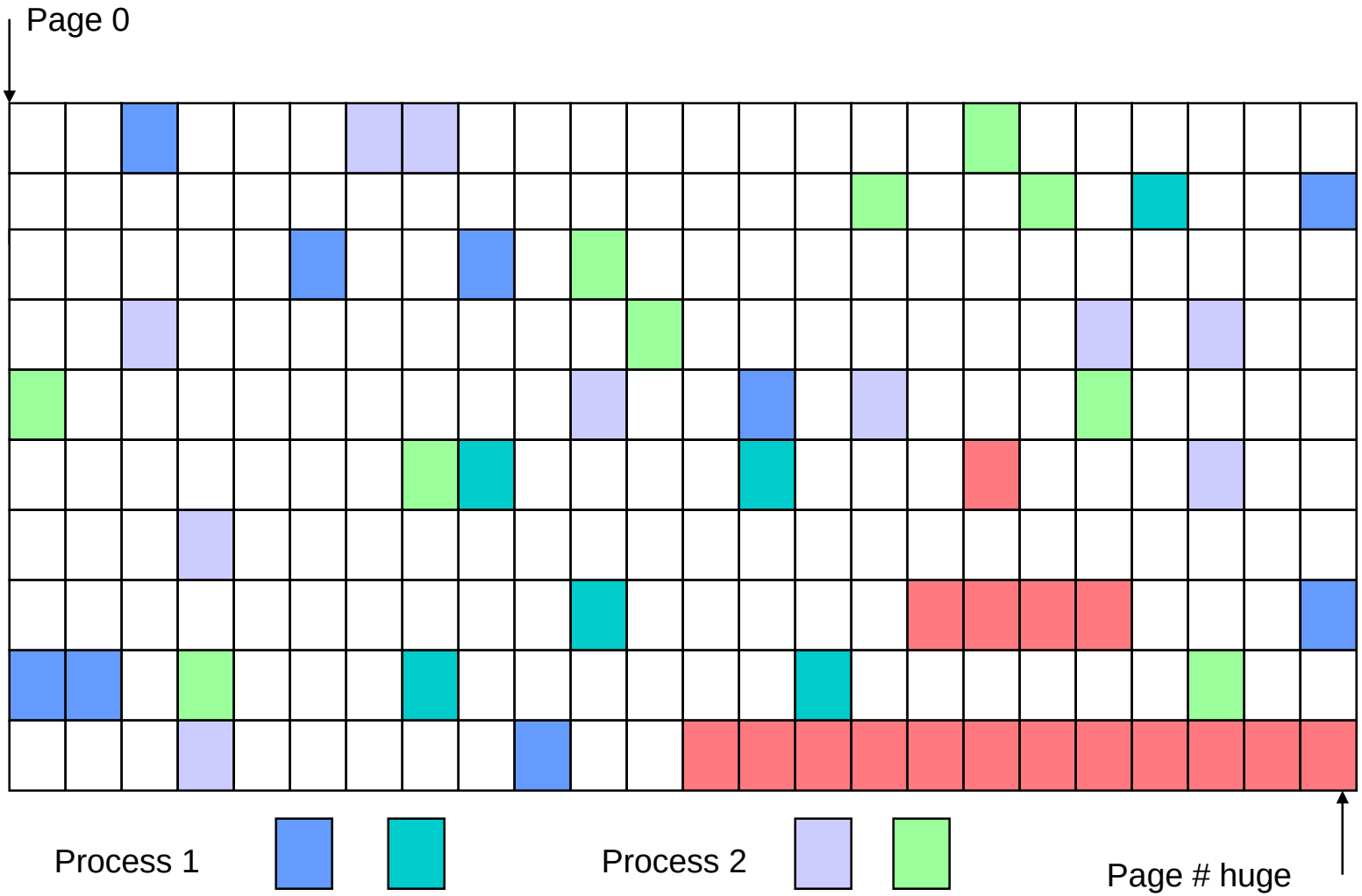
VMS Virtual Addressing



Mapping Process Pages

- Pages of process VA are mapped onto the machine's physical memory
- All address ranges for all processes from 2 GB to 4 GB are mapped onto the same physical pages
- The following slide notes this mapping for two processes

VMS Memory Mapping



What is in VMS System Space?

- A number of control structures to manage VMS
  - Process page tables
  - Stacks
  - Non-paged memory
  - Reserved for non-paged memory growth [NPAGVIR]
  - Other Stuff
  - Unassigned space
    - This is what Rdb can use for its work

What is in VMS System Space?

- Seen with SDA
- SDA> clue memory/layout

```
System Virtual Address Space Layout:

-----------------------------------

   Item                                  Base              End          Length

System Virtual Base Address       FFFFFFFE.00000000
PFN Database                      FFFFFFFE.00000000  FFFFFFFE.01400000  01400000
Permanent Mapping of System L1PT  FFFFFFFE.01400000  FFFFFFFE.01402000  00002000
Global Page Table (GPT)           FFFFFFFE.01402000  FFFFFFFE.015CCC40  001CAC40
Lock ID Table                     FFFFFFFF.7FA08000  FFFFFFFF.80000000  005F8000
etc.
Nonpaged Pool (initial size)      FFFFFFFF.81850000  FFFFFFFF.86E24000  055D4000
Nonpaged Pool Expansion Area      FFFFFFFF.86E24000  FFFFFFFF.9366A000  0C846000
etc
398 Balance Slots - 27 pages each FFFFFFFF.93884000  FFFFFFFF.98C78000  053F4000
Mount Verification Buffer         FFFFFFFF.9A582000  FFFFFFFF.9A584000  00002000
Demand Zero Optimization Page     FFFFFFFF.9A584000  FFFFFFFF.9A586000  00002000
Executive Mode Data Page          FFFFFFFF.9A586000  FFFFFFFF.9A588000  00002000
System Space Expansion Region     FFFFFFFF.AA000000  FFFFFFFF.FFDF0000  55DF0000
etc.
SDA>
```

What is in System Space?

- One needs to turn these hex numbers into something more intuitive, decimal

```
System Virtual Address Space Layout:
-------------------------------------
   Item                                    Base              End          Length, Decimal

System Virtual Base Address          FFFFFFFE.00000000
PFN Database                         FFFFFFFE.00000000  FFFFFFFE.01400000
Permanent Mapping of System L1PT     FFFFFFFE.01400000  FFFFFFFE.01402000
Global Page Table (GPT)              FFFFFFFE.01402000  FFFFFFFE.015CCC40
Lock ID Table                        FFFFFFFF.7FA08000  FFFFFFFF.80000000      6,258,688
etc.
Nonpaged Pool (initial size)         FFFFFFFF.81850000  FFFFFFFF.86E24000     89,997,312
Nonpaged Pool Expansion Area         FFFFFFFF.86E24000  FFFFFFFF.9366A000    210,001,920
etc
398 Balance Slots - 27 pages each    FFFFFFFF.93884000  FFFFFFFF.98C78000     88,031,232
Mount Verification Buffer            FFFFFFFF.9A582000  FFFFFFFF.9A584000
Demand Zero Optimization Page        FFFFFFFF.9A584000  FFFFFFFF.9A586000
Executive Mode Data Page             FFFFFFFF.9A586000  FFFFFFFF.9A588000
System Space Expansion Region        FFFFFFFF.AA000000  FFFFFFFF.FFDF0000  1,440,677,888
etc.
SDA>
```

VMS Use of S0 Address Space

- VMS engineering has been working on moving structures out of S0 address space
- Structures moved to S2 space, 64-bit addresses
  - Lock management structures
  - Large process-related structures
- Requires VMS 7.1 and 7.2

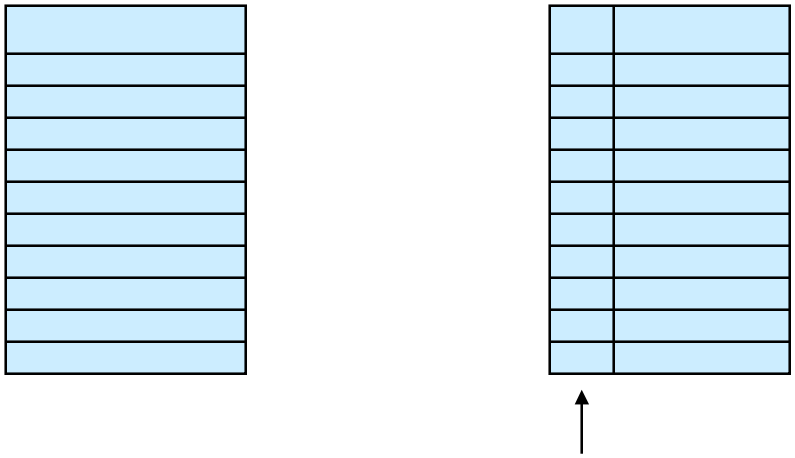
VMS Tuning Gotchas

- NPAGVIR is often set by AUTOGEN to 4 times NPAGEDYN.
  - In large memory systems NPAGDYN is often also set very large
  - In large memory systems this becomes the major fraction of system VA
- For large memory systems AUTOGEN initially sizes for hundreds or thousands of processes
  - And those processes are often sized for humongous virtual addresses
- The net result is that VMS takes over the 2 Gb of VA that has to be shared with Rdb
- The AUTOGEN daemon must be tamed
  - By creating appropriate MODPARAMS.DAT entries
- Warning … If the system runs out of non-paged memory it will stop … abruptly

What is a Row Cache?

Management Structures

The Detailed Cache Entries



24 Byte/entry overhead per slot

What are the Management Structures?

- From the designer's point of view, the most interesting thing is the presence of a "hash table"
  - Stores the DB-keys of rows in the table
  - Lookups of DB-keys are via hash algorithm
    - Sort of biased to smaller number of lines on a page
  - Point at the slot # of that row in the cache
  - Is an index
- Hash tables are always sized large
  - Rounded up to 2^N entries, such that the result is just larger than the number of slots designed for a cache
  - Caches with 30,000 slots will have a hash table of 32768
  - Caches with 32,000 slots will have a hash table of 32768
  - Caches with 33,000 slots will have a hash table of 65536
- Hash tables can be extremely large when dealing with caches with millions of entries
  - Can cause trade offs when managing the plan for S0 address space

How the Cache is Accessed

- Access to rows in the cache is by DB-key
  - That is, hash the DB-key and then access the slot
- A sequential scan will read database page to get the DB-keys
  - Then access data in the cache to get the latest version of the row

Efficiencies in Using a Row Cache

- Page locks disappear from processes, except during inserts
  - Space must be reserved for new line
- I/O for update of updated objects is transferred to the Row Cache Server [RCS] process
  - If snapshots are in effect, processes still update the database page tail to update snapshot information
    - Often results in processes writing the updated rows back to the database in piggy-back fashion
    - Result will be that the RCS process has little to do at checkpoints.

Requirements For Cache

- Single database node [for the moment…]
  - No synchronization of memory as in global buffers
- Fast Commit
  - RCS process can checkpoint to backing store or to database
  - Backing store checkpoints make sense only if snapshots are not active [disabled or enabled deferred]
    - Because, when object updated, snap page pointer must be updated on live data page

Cache Parameters

- There are just a few parameters that describe each cache
- Cache width, the size reserved for each row
- Cache length, the number of rows in the cache
- The number of windows [VLM cache only]
  - Default is 100
  - Minimum is 11
  - It is our belief that one should select values toward the minimum
- The number of slots reserved by processes when writing to the cache
  - So each process doesn't have to repetitively allocate entries when writing
- How many slots are swept at a time

Where Does the Cache “Live”?

- Caches can be configured in several ways:
- Management structures
  - System space [S0]
  - Global section
- The detailed entries
  - VLM mapped by window in process space
  - System Space
  - Global section in process space
- Because one is usually interested in very large caches configuring them in a process global section will lead to the same memory management/working set/page fault problems that exist with large process global buffer pools
  - We don’t do it

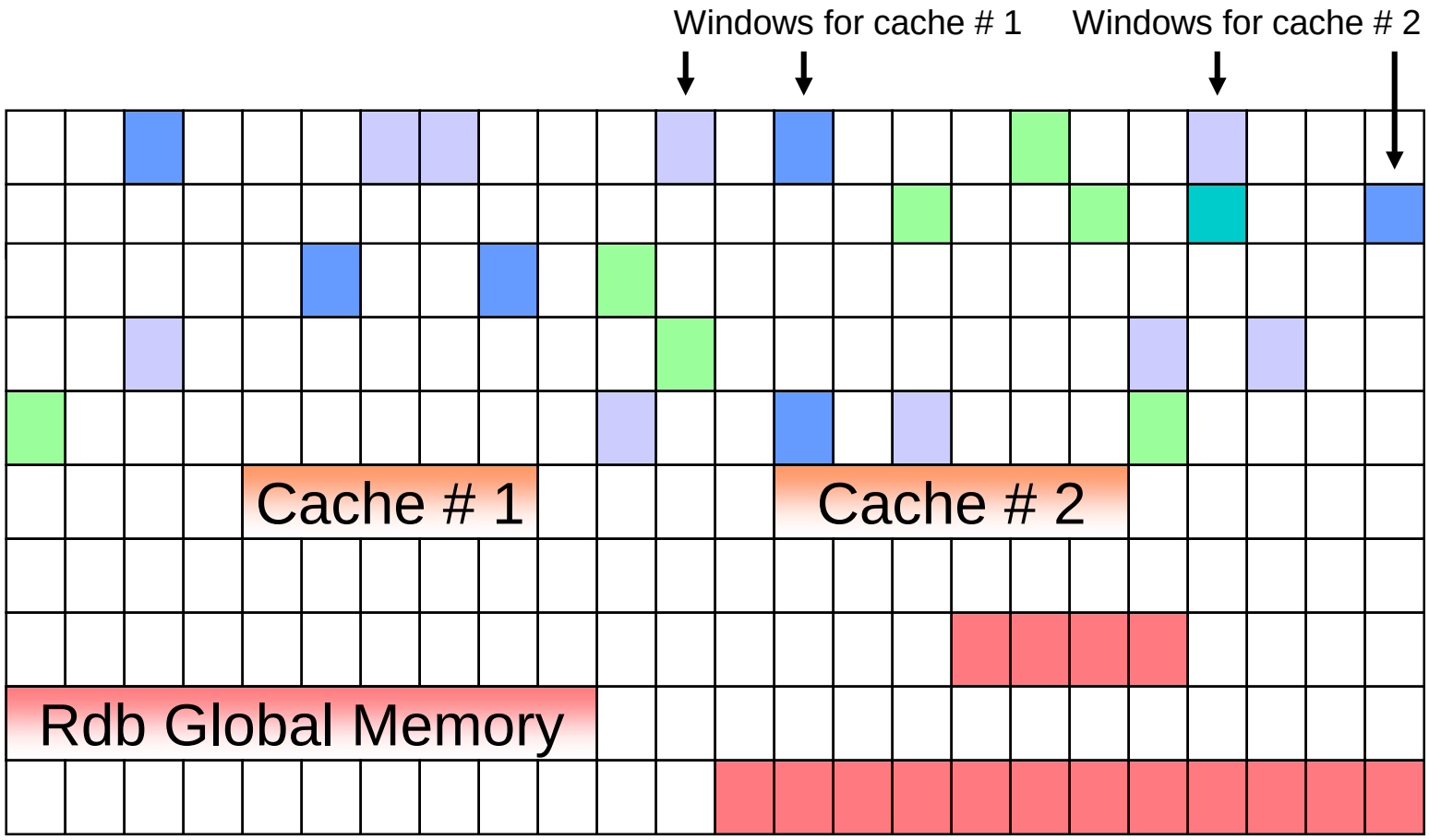
System Space Caches

- Control structures live in in system space
- The cache entries also live in that same global section
- Use for smaller caches
  - Because system address space is limited
  - May want to also use for global buffer pool
- All parts of the cache are visible to all processes simultaneously

VLM Caches

- Control structures live in in system space
- The cache entries are stored in physical memory “stolen” from VMS [removed from VMS’ memory management control.]
- Specific pages of process P0 address space are used as “windows” into the VLM
  - The page table entries for these pages are manipulated by Rdb to point at entries in the VLM section
  - Window turns describe re-mapping of a window to a different set of rows
    - Not free, but not all that expensive
  - It is “nice” to have rows live in only 1 window
    - suggests sizing criteria for cache widths

VLM Caches



Another Dimension to Caches

- Caches may be used to map physical areas or logical areas
- A single physical area cache will store data from one or several physical areas
  - Could cache all B-tree index nodes of same size in a single large physical cache
  - Could cache, for instance, RDB$SYSTEM with multiple tables and indexes in single physical cache
    - Must deal successfully with the variance of lengths of different objects

Another Dimension to Caches

- A single logical area cache can collect horizontally partitioned objects into a single cache
  - Works by matching logical area name in the area inventory
- Could also match other incidentally identically-named objects such as system records for mixed areas
  - Could do weird things if index and table names are the same

What Is Not Cached?

- Rdb will not cache certain objects
- If an object (row, B-tree node, hash bucket) is fragmented it will not be cached
- Instances of objects that are too wide to fit in the cache
- AIP, ABM and SPAM pages are not cached [yet]
  - This is one area where global buffers will be a big help

Designer's Decisions

- Memory is expensive and precious and should be effectively deployed
- The cache designer is then faced with the following decisions
- What database objects should be cached
- How large [length, width] should the cache be?
- Where should cache be located, System or VLM
- What kind of cache should be used, logical or physical
- How many caches are needed

Caching Narrow Rows

- Caching very narrow tables can be extremely frustrating
  - Minimum cash width is 16 bytes
  - Add to this another 24 bytes of overhead
  - Effectively caching a 12-byte row will consume 40 bytes of effective memory, not a good payback
- Leads to some interesting approaches to query tuning
  - Consider indexing the entire record, perhaps in several ways
    - Prefix compression of index may lead to good density
    - Could create concurrency problem
  - Cache the index
  - Write outlines that force the use of the index
  - Cache the table by caching the index(es)

Which Objects Should Be Cached

- Obviously one wants to cache objects which are frequently accessed
- Indexes
  - Duplicates in the B-tree and hashed indexes create lots of small-sized objects.
    - Space wasted in slots used to cache these objects
    - Ranked indexes are not subject to this problem
    - Or add columns to index to make unique
  - Hash indexes can be a problem because of variable bucket sizes
- Code tables, but they might be cached by just indexing the entire table
- Important tables

What Should Not Be Cached

- You cannot create a humongous cache with 500 million rows
  - There will not be sufficient virtual address space in S0 to handle the cache hash table
  - VMS limitation
- It doesn't make a lot of sense to cache objects that are read transiently and seldom
  - Such as large hash indexes or placed tables in mixed format areas
  - Such objects are seldom revisited so caching would have no benefit

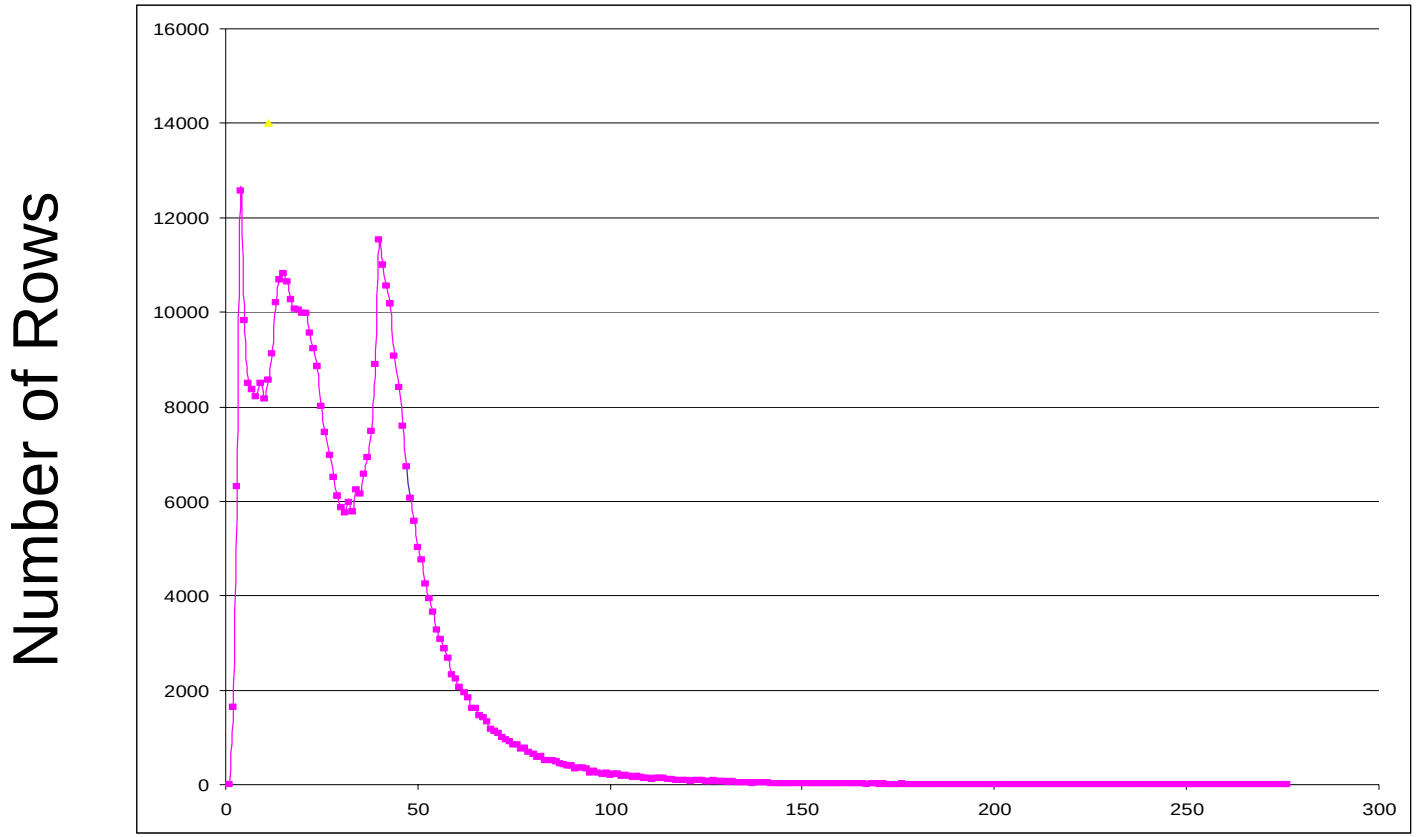
How Wide Should The Cache Be?

- Width of the cache is determined by the width of what is being cached
- For tables is row size, for index is index size
  - B-tree max size is unambiguous, except for duplicates nodes
  - Hash buckets, you have to do an analysis
- We have tended to disable compression on tables being cached
  - Don't want to play the compression statistics game
  - Trade memory for speed
- Standardize B-tree index node sizes, favor ranked indexes
  - Do *not* size B-tree nodes to fit on database pages
    - You can't completely ignore the page size, of course…do not cause nodes to fragment!
    - Size pages to comfortably hold nodes
  - Do size B-tree nodes to fit on alpha pages (remembering the 24-byte overhead)

How Wide Should Cache Be?

- If you have a table with compressed rows, you need to pick a width that is appropriate to cache some desired fraction
- Do an RMU/DUMP of the area containing the table into a file
- Do a search to obtain a list of line lengths, put this into a file
- Edit out the junk leaving only the line lengths
- Import into eXcel
- Generate frequencies based on predetermined intervals
- Plot a graph, see the next slide

How Wide Should a Cache Be?



How Long Should a Cache Be?

- If one wants to size a cache to store “everything” it is pretty easy
  - For tables, `select count(*) from table` and size appropriately
  - For indexes, `RMU/ANAL index/option=full`
    - Will generate statistics on numbers of nodes/buckets
    - Remember B-tree indexes grow logarithmically as the associated table grows
    - Size appropriately
- Remember to allow for growth of object
  - Determined by application
  - Results in caches which are always a bit long

Cache Thrashing

- If a cache has too few slots to cache the objects designated for it, the cache will thrash
- Entries will continually be discarded and new ones entered
- Effectiveness of this cache is diminished, perhaps to the point of no effectiveness
- Rule: be wary of caches which thrash
  - Sometimes you have special knowledge that indicates that thrashing is o.k.
    - E.g. data enters db, is processed for a while and then becomes somewhat archival
  - Be especially careful of caches containing indexes which thrash

Thrashing Cache

```
Node: XXXX (1/1/1)      Oracle Rdb V7.0-5 Perf. Monitor  25-SEP-2000 16:46:08.27
Rate: 0.50 Seconds              Row Cache Status            Elapsed: 06:37:59.81
Page: 1 of 1                     Sample Database                   Mode: Online
-------------------------------------------------------------------------------
                        For Cache: sorted_index_S_01
Statistic.Name Stat.Value Percent

Total slots:        4500  100.0% Slot Length: 488  Hash slots: 8192
Slots full:         4322   96.0% Use:         19   0.4%
Slots empty:         178    3.9% Rsv:         23  12.9%
Marked Slots:       2067   45.9% Hot:         17   0.8% Cold:       2050  99.1%
Clean Slots:        2433   54.0% Hot:          0   0.0% Cold:       2433 100.0%
Used Space:         2074k  94.4%
Wasted Space:         34k   1.5%
Free Space:           86k   3.9%
Hash Que Lengths:  Empty:3870 1:4322     2:0        3:0        4+:0
Cursor position:   3595 of 4500 wrapped 395 times ← The cache is thrashing a bit
Cache latched:     No
Cache is full:     No
Cache modified:    Yes
Section name:      RDM70R_0D2MTI2D (18 pages each 8192 bytes)
VLM Section name:  RDM70V_0D2MTI2D
Number of checkpoints: 1867 Last: 25-SEP-2000 16:35:16.20 Location: 3239:536329
Cache Recovery:    3240:5521
```

What Happens if Cache is Full

- If a cache is full and a process wants to add entries, but all entries in the cache are marked [have been updated] the cache is “clogged”
- Process requests that RCS process sweep the cache
  - Rows are swept if they have been modified thus freeing slots for replacement
  - A limited number of rows are swept [the sweep count for the cache]
  - This value should be large enough to permit a job to complete with minimal numbers of sweeps

Management Problems

- When tables change, the associated caches should be re-examined
- Rows may no longer fit in cache if more columns are added (cache is too narrow)
  - Since Rdb does not maintain the logical area size in the area inventory, resizing a cache requires careful tracking of what is changed & don't forget the null-bit-vector
  - *Rdb will automatically size the width of a cache if it is not specified at cache definition time.*
- Cache may begin to thrash if more instances are added (cache is too short)

VLM or System Cache

- The answers are generally easy
- If the cache is small map it to system space
- If the cache is large, use VLM
- Can see estimated memory requirements for a cache through RMU/DUMP/HEADER=row_cache/out=<file>

Memory Requirements – Interpretation

```
Shared Memory...
     - System space memory is enabled (OpenVMS Alpha only)
     - Large memory is enabled (OpenVMS Alpha only)
     - Large memory window count is 100
   Cache-size in different sections of memory...
     - Without VLM, process or system memory requirement is 91,267,072 bytes
     - With VLM enabled (OpenVMS Alpha)...
         - Process or system memory requirement is 8429568 bytes
         - Physical memory requirement is 82838452 bytes
         - VLM Virtual memory address space is approximately 1643048 bytes
```

- The number of windows is excessive for this cache, yes, it is also the default
- If a process-based global section were used it would require 91,267,072 bytes (sum of control structures and data)
- The windows require a process VA of 1,643,048 bytes
- Control structures require 8,429,568
- 82,838,452 bytes are used to cache the rows

Logical or Physical Caches?

- Logical area caches trump physical area caches
  - If a table/index can be cached both ways, it is cached in the logical are cache
- One goal is to get most effective use of the total physical memory available
- Consider tailoring each cache to each database object
  - Can result in large numbers of logical area caches
- Cache RDB$system storage area in single physical area cache. Use distribution technique described earlier
  - RDB$database logical area is extremely wide
  - If you wish, cache using a 1-row logical area cache

Logical or Physical Caches

- Large numbers of caches can be a management headache
- Consider pooling all B-tree indexes into a small number (1, 2, 3) caches as determined by node size (2,024, 1000, 488, 232 are node sizes that will cause slots to fit within a single alpha page)
  - Keith likes 1000 byte nodes on 4 block pages
  - Still have to worry about locking contention
- If you want to cache segmented strings (lists) use physical area caches
  - Default Rdb list area is a candidate
  - Requires size distribution analysis

How Many Caches

- The rational answer is as many as needed
- However, too many caches can lead to management problems
  - Most caches are oversized so each has some amount of wasted space
  - Can’t see everything on a single RMU screen, even if 48 lines long
- Suggests pooling caches of similar width
  - Will make it difficult to determine which object is causing cache to thrash

Watching the RCS Process

- Define the logical name [at the system level]
  - RDM$BIND_RCS_LOG_FILE = RCS_PID.LOG
- This will cause the RCS process to log its activity into a log file
- File is named RCS<pid>.log And stored in the same directory as the database root file
- Allows DBA to examine frequency and duration of checkpoints on a cache by cache basis
- Can also see sweeps that occur
- Cause RCS to reopen log file by defining one or both logical names
  - RDM$BIND_RCS_LOG_REOPEN_SECS
  - RDM$BIND_RCS_LOG_REOPEN_SIZE

Managing Database Startup

- We set all production databases to open is manual
  - Starting/stopping row cache server, creating/deleting sections is a big overhead
  - If one is pegging VMS memory resources you do not want to introduce fragmentation.
    - We have seen cases where one must reboot VMS to be able to open the database a second or third time if database parameters are changed.
- Cause the RCS process to start when the database is opened
  - Define RDM$BIND_RCS_CREATION_IMMEDIATE 1
- Cause all caches to be instantiated immediately
  - Allows all memory to be claimed by database
    - Necessary to ensure contiguous space is available for caches and their associated global structures
  - Define RDM$BIND_RCS_INITIAL_MAP_ALL_CACHES 1
- Have to introduce waits in VMS startup because RMU/OPEN returns almost immediately

Managing Checkpoints

- Row Cache requires that fast commit be enabled
- Result is that one must be concerned about managing fast commit checkpoints
  - Journal growth
  - Wall clock time
  - # of transactions
- Executed in the context of the Rdb commit/rollback code
  - If an attach does no work, it won't checkpoint!
  - Could result in very unpleasant DBR durations
  - Can be forced by RMU/checkpoint
    - Can cause your I/O subsystem to seize-up for a loooong time

Summary

- Benefit: Row caches can significantly costs of running an application
  - Massive I/O reduction possible
  - Significant page lock contention can be eliminated
- Has been a silver bullet for certain performance problems

Summary

- Costs:
  - Fast commit context. Idle processes attached to the database can extend recovery time after process or system failures
    - RCS sweeps and checkpoints can result in extremely large I/O bursts to disk if in heavy update environment
    - Need to be concerned about I/O subsystem design
  - Significant amounts of physical memory
  - A single VMS node
  - Careful tuning of VMS
  - Design effort
    - If coupled with a large global buffer pool can allow designer to focus cache efforts on a limited number of effective issues
      - Allow “everything else” to be handled in the buffer pool
  - Continued DBA attention to cache parameters, nothing comes free

Thanks Rdb Engineering!

The authors want to thank Rdb engineering for their close cooperation and support while we were trying the new Row Caching code and …