I/O is a fundamental characteristic of Db2 performance. It is not only important to understand I/O access patterns via monitoring, but also to be able to tune Db2 prefetching and I/O. Optimizing Db2 I/O is critical both for row- and column-organized table performance. Attend this session to peek behind the curtain at Db2 prefetching and I/O both from a functional and monitoring perspective and learn tips and tricks for tuning Db2 to maximize I/O throughput.

Please note :

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice and at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Notices and disclaimers

•U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

•Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

•IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

•Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

•Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

•References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

•Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

•It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

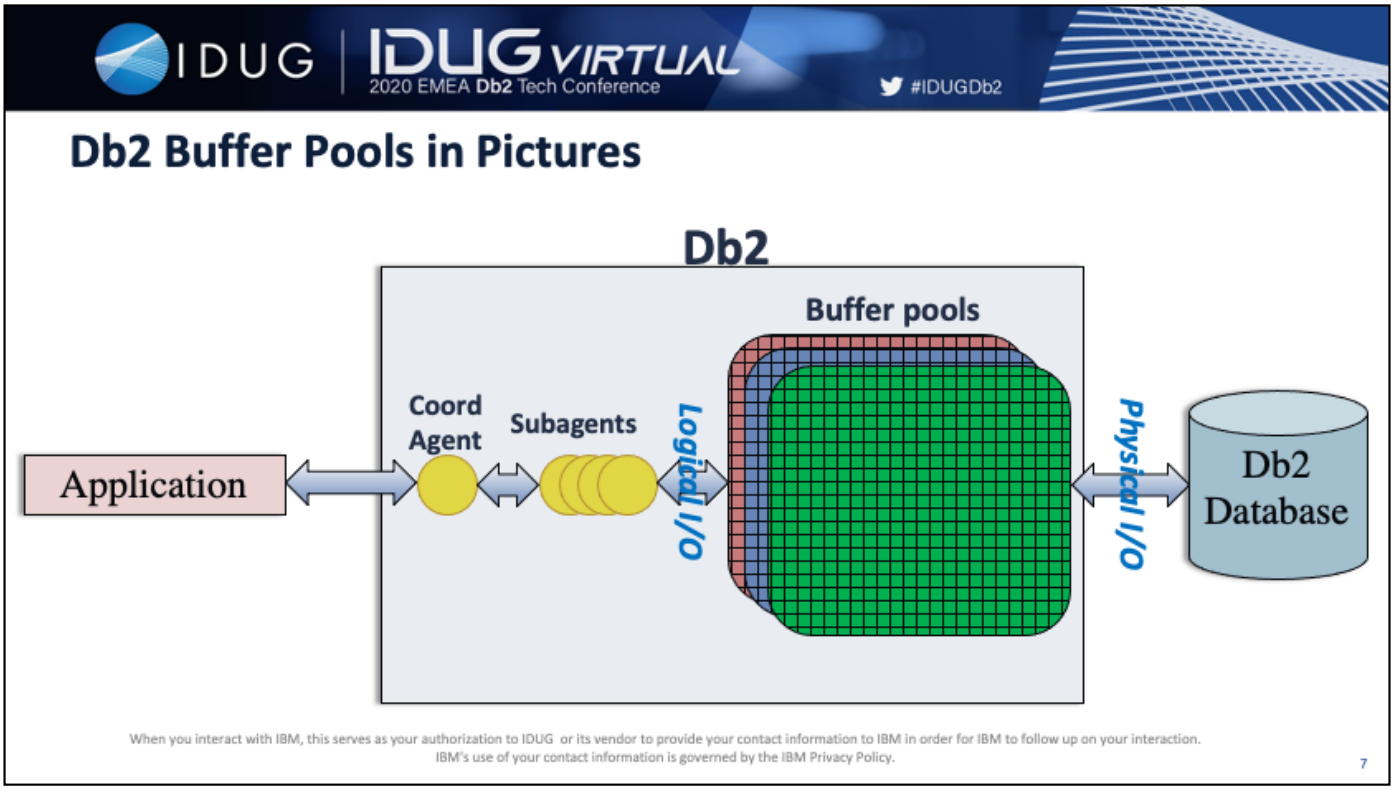
Agenda

- I/O and Buffer Pools
- Prefetching
- Page Cleaning
- Some Key Tuning I/O Tips, Tricks, and Best Practices

I/O and Buffer Pools

I/O – What's it All About?

- I/O (Input/Output) is a fundamental characteristic of reading/writing data to and from disk
- Accessing data on disk (SSD, spinning disks, etc.) may vary in terms of performance, but is not free
- Optimizing data access via buffer pools is a fundamental feature of Db2
- Proper buffer pool and related I/O configuration is critical to optimize workload performance



The application opens a connection to process a statement that is initially handled by the Coordinator Agent. This agent may subdivide the task across multiple subagents. Depending on the application request, we may or may not execute logical I/O to check the buffer pools for pages. If a logical I/O is executed, and the desired page does not already exist in the buffer pool, a physical I/O will take place to fetch the page from disk. It then resides in the buffer pool until it is later victimized to make room for another page.

Buffer Pool Basics (1|2)

- A buffer pool is an area of main memory that has been allocated by the database manager for the purpose of caching Db2 data pages (not including LOBs and LFs)
- Every database must have a buffer pool and every table space must be assigned to a buffer pool
- A buffer pool caches the pages of one or more table spaces of the same page size
- Buffer pool definitions can be found in the SYSBUFFERPOOL catalog table

LOBs may be cached in the buffer pool if they have been inlined in the table, in which case they exist in normal data pages

Buffer Pool Basics (2|2)

- There is one default buffer pool (IBMDEFAULTBP) created when the database is created
- In MPP buffer pools exist on each database partition and are independently sized
- In pureScale a local buffer pool (LBP) exists on each member and a group buffer pool (GBP) exists in the CF

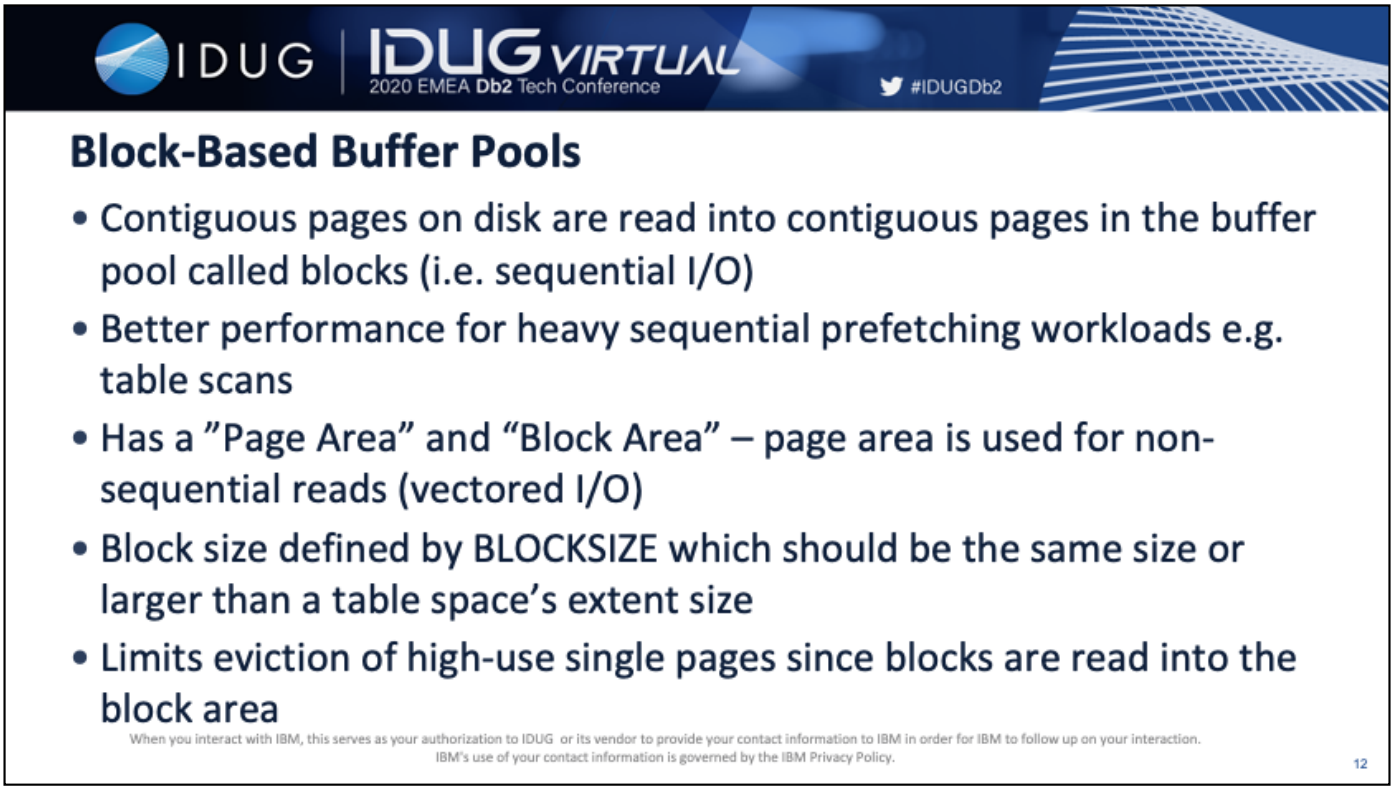
- IBMDEFAULTBP has a page size of 4K by default unless a registry variable implicitly changes it (i.e. DB2_WORKLOAD=ANALYTICS changes the default to 32K)
- A buffer pool can be created in a specific database partition group so it may not be defined on all nodes

Buffer Pool Attributes

- A buffer pool can have a page size of 4, 8, 16, or 32K – same as a table space
- A buffer pool's size is defined as a number of pages of the buffer pool's page size
  - A buffer pool can be set to be automatically resized by STMM
  - A buffer pool's size can be altered dynamically (IMMEDIATE) or be deferred until the next activation (DEFERRED)
- There are two types of buffer pools

Page-Based Buffer Pools

- This is the default type of buffer pool
- Contiguous pages on disk are read into non-contiguous pages in the buffer pool (i.e. scattered or vectored I/O)
- All buffer pool pages are in the “Page Area”



- Block-based buffer pools are not available in pureScale
- Block area can be no larger than 98% of the buffer pool
- The block size should be the same as or larger than the tablespace’s extent size or block space will be wasted. If too much is wasted, Db2 may decide to revert to using the page area. 50% of the block is acceptable to waste before considering vectored I/O.

Buffer Pools – Into the Deep End! (1|2)

- Every buffer pool consists of three main parts:
  - Page array
  - Buffer pool Page Descriptor (BPD) array
  - Hash Table
- These structures reside in the buffer pool heap
- The page array is the memory that stores the actual Db2 data pages

Buffer Pools – Into the Deep End! (2|2)

- The BPD array is the memory that holds BPDs – the metadata structure that holds info about a given page (i.e. page key, dirty state)
  - There is a 1-to-1 relationship between pages and BPDs
  - Contains a pointer to the page for which the BPD is associated
  - Contain the page/BPD latch
- The Hash Table is used for quickly looking up a BPD for a given pageKey
  - A pageKey is a unique identifier for a page (tablespace ID, object ID, object type, object page number)

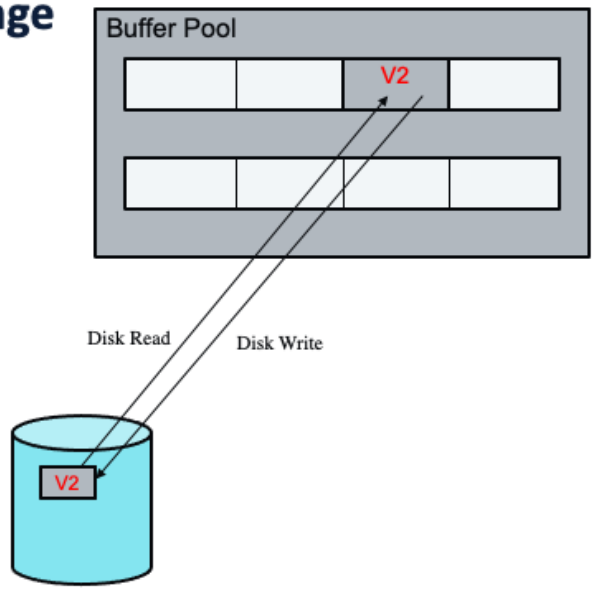
Multiple levels of hashing are used along with memory optimizations to maximize performance.

How are Buffer Pools Used?

- When data in a table is first accessed, the page that contains that data is read from disk into the buffer pool via physical I/O
- Pages remain in the buffer pool until the database is deactivated or the buffer pool slot is required for a different page (victimization)
- Page access is controlled via a BPD latch (page latch)
- Buffer pool hit ratio is very important!
- Pages can be modified (dirtied) in the buffer pool and can have contents different than that on disk

Buffer pool hit ratio is a measure of how often a page request finds the page in the buffer pool without having to read the page from disk

Example of "Fixing" a Page



Pages – A Love/Hate Relationship

- Pages in the buffer pool have "weights"
  - Hints from the previous fixers of a page about whether they anticipate needing the page again
- There are 3 page weights:
  - Hated: fixer will not need the page again
  - Unloved: fixer is unsure if it will need the page again
  - Loved: fixer will need the page again

Weights age over time. We favor Loved pages, but can't love them forever.

Page Victimization

- When a fixer needs to read a page from disk into the buffer pool it must choose a slot to read the page into
- An ideal slot is one that is either empty/not-in-use or one that contains a hated and clean page
- We track dirty and hated/clean pages to speed up this process

Synchronous vs. Asynchronous I/O

- Synchronous (physical) I/O is expensive
- Asynchronous I/O is very powerful!
  - E.g. Prefetching and Page Cleaning
  - Able to greatly reduce I/O costs

Prefetching

Prefetching Basics

- In many situations an agent can anticipate which pages are going to be needed and request that they be read into the buffer pool before they are needed
- Prefetchers are background EDUs that service agents' prefetch requests
- The optimizer chooses what type(s) of prefetching to enable for an ISCAN-FETCH or index scan operation

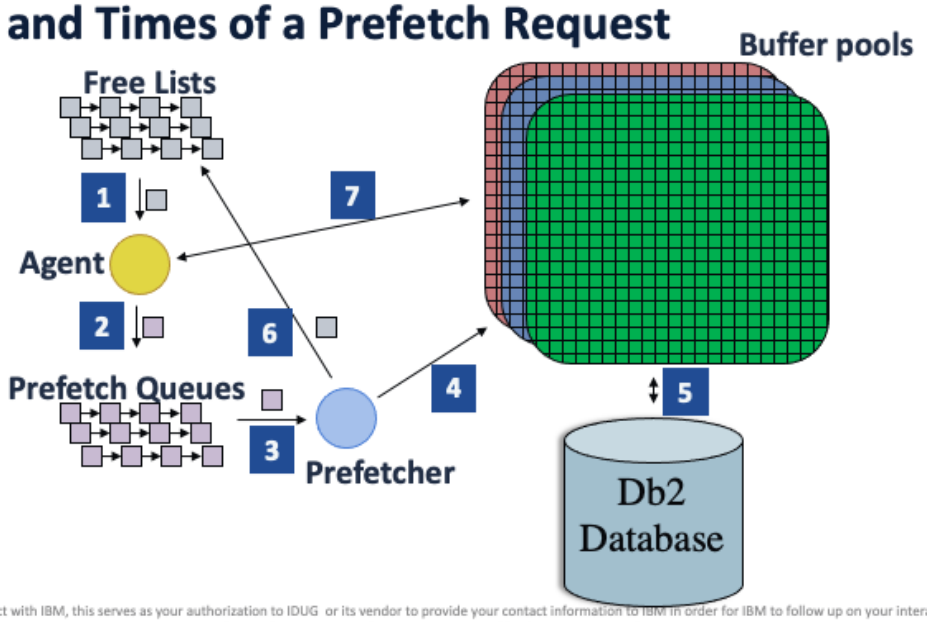
Prefetch Architecture

- There are two important prefetch structures:
  - Free Lists
  - Prefetch Queues
- Typically there are the same number of Free Lists and Prefetch Queues
- There is a fixed amount of prefetch requests created at database activation – typically just over 10,000
- The number of prefetchers is controlled by the NUM_IOSERVERS database config parameter
- The table space config PREFETCHSIZE controls how many pages agents request at once, with some exceptions

Free Lists: linked lists of empty requests that can be ”filled out” by an agent

Prefetch Queues: linked lists of “filled out” requests that need to be fulfilled. This list is FIFO

The Life and Times of a Prefetch Request

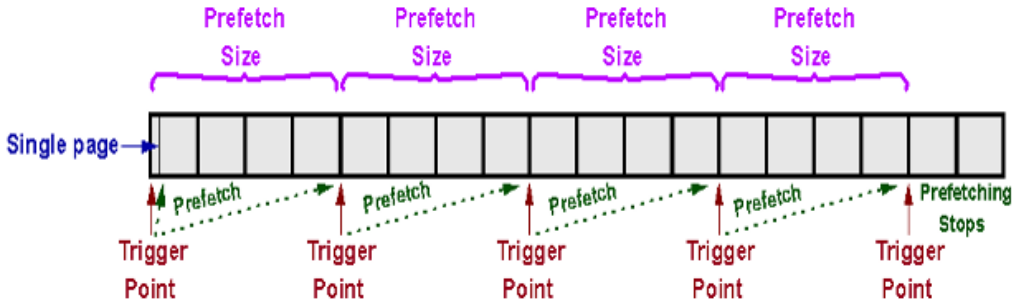


1. Agent locates an available prefetch request from a free list
2. Agent populates the prefetch request with info about what pages to prefetch and places it on a prefetch queue
3. Prefetcher retrieves the populated prefetch request from a prefetch queue
4. Prefetcher drives logical I/O for requested pages
5. Physical I/O takes place for any pages that are not already in the buffer pool
6. Prefetcher resets the prefetch request and places it back on a free list
7. Agent executes logical I/O for the prefetched pages

Primary Prefetch Categories

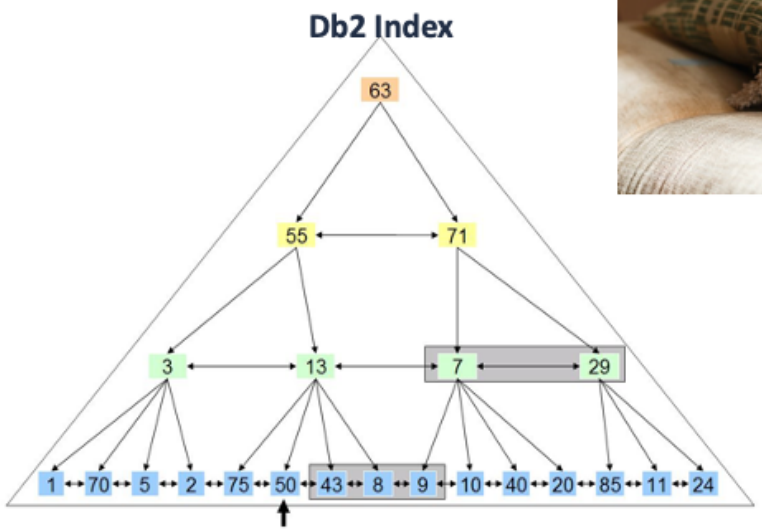
- Sequential prefetching
- Readahead prefetching
- List prefetching
- Dynamic list prefetching

Sequential Prefetching



Sequential prefetching works well for table scans since the page numbers would be in order, index scans on nicely-organized indexes, and while accessing data pages of a clustered index during an ISCAN-FETCH operation.

Readahead Prefetching

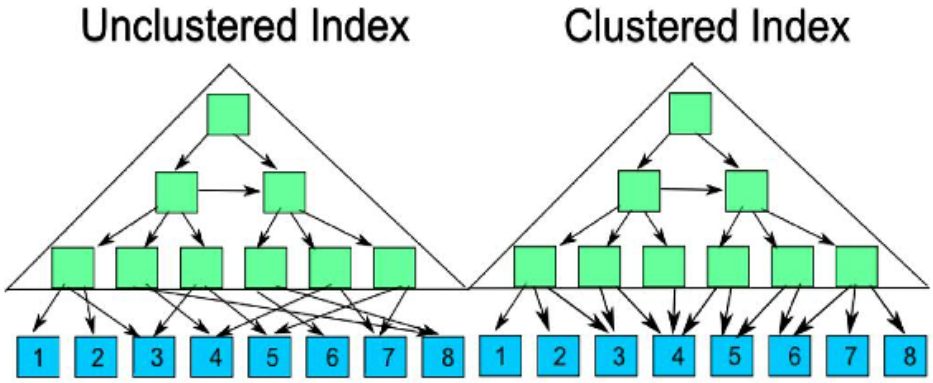


Readahead prefetching is used in two cases at the optimizer's discretion):

- Index scans on disorganized indexes
- Accessing data pages for an unclustered index for an ISCAN-FETCH operation

Sequential detection plus readahead prefetching together form smart index and smart data prefetching.

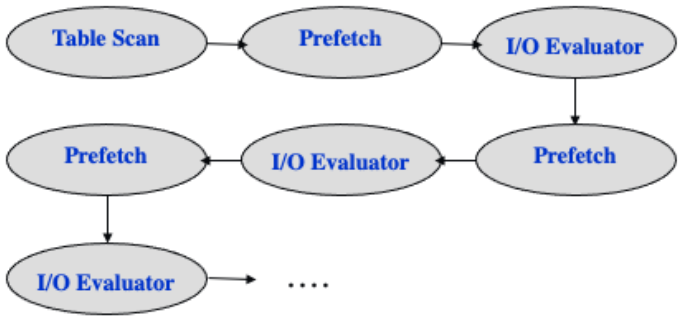
List Prefetching



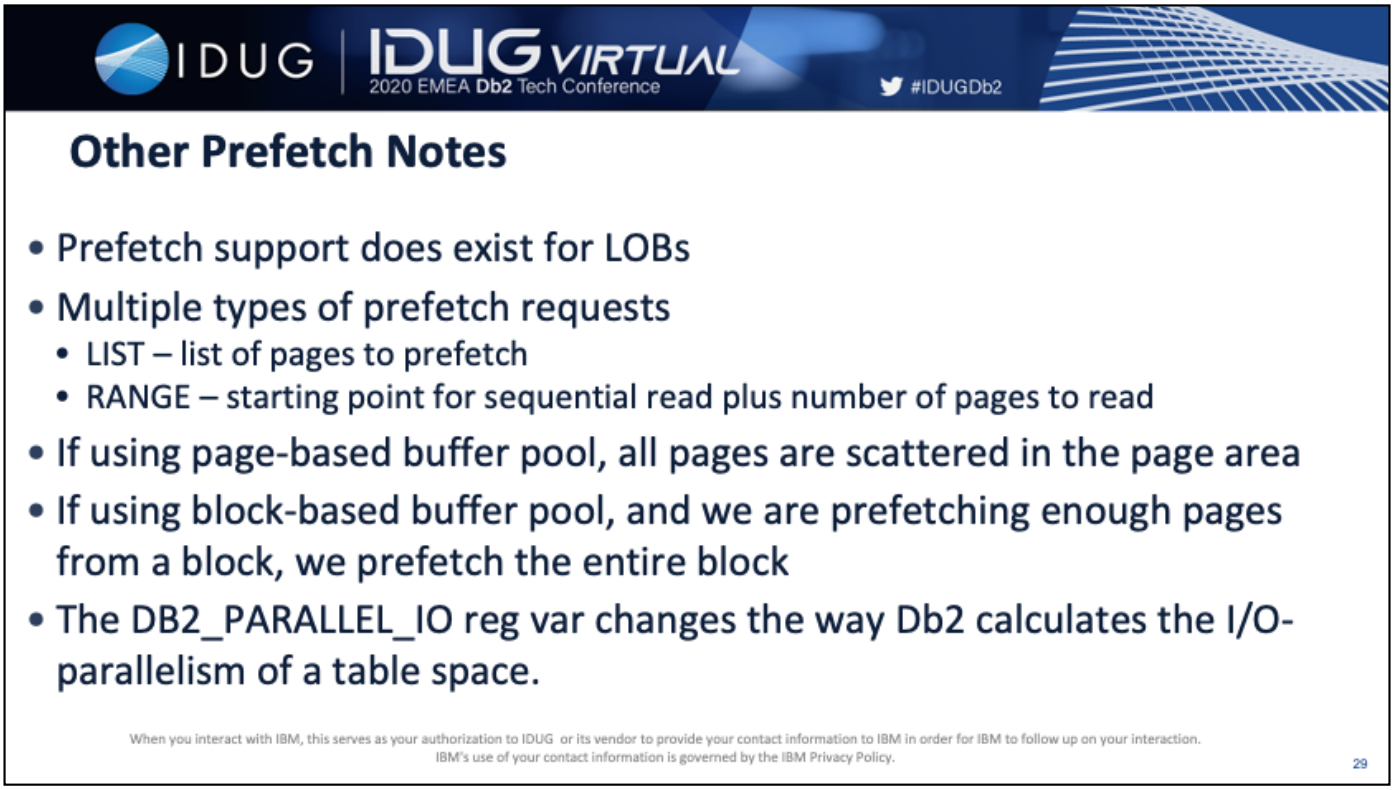
Select * where name between 'A' and 'I'

Depending on available statistics, the optimizer may choose a RID-LIST plan instead of an ISCAN-FETCH for unclustered indexes. This means that the index scan alone takes place first, and then the qualifying data page RIDs are sorted before those pages are accessed. After the sorting phase, we are able to prefetch the appropriate data pages as needed using list prefetching.

Dynamic List Prefetching – BLU only



Dynamic list prefetching is specific to BLU tables. The idea of this technique is that the BLU query runtime engine may have multiple threads executing evaluation chains. These chains are composed of multiple evaluators that each perform one operation similar to regular plan operators. Multiple threads can work on a table such that they perform a straw scan to determine which section of the table to process. Once that happens, a work unit corresponding to that section of the table flows through the thread's evaluation chain from evaluator to evaluator. The TSNs in the work unit are filtered out as it progresses. Since large portions of the table may be filtered, prior to every I/O evaluator we prefetch only the data pages for the column that will be required for the subsequent I/O evaluator. We then attempt to hold those work units in the Prefetch evaluator while other work is done until the prefetched pages are available in the buffer pool, thus minimizing physical I/O while avoiding prefetching unneeded pages.



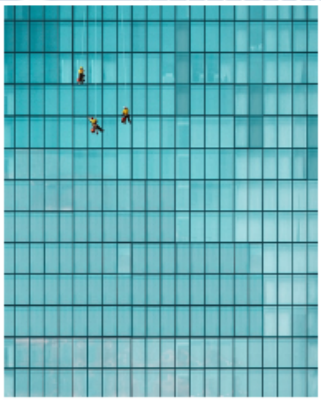
LOBs may be prefetched into memory buffers instead of into the buffer pool.

LIST request -> vector or block I/O – depends on if we are using a block-based buffer pool, and a block is available for victimization. Also, we must be prefetching at least 50% of the pages from the block.
RANGE -> vector or block I/O – depends on if we are using a block-based buffer pool, and a block is available for victimization.

DB2_PARALLEL_IO enables multiple prefetch requests to be driven over table spaces containing multiple containers.

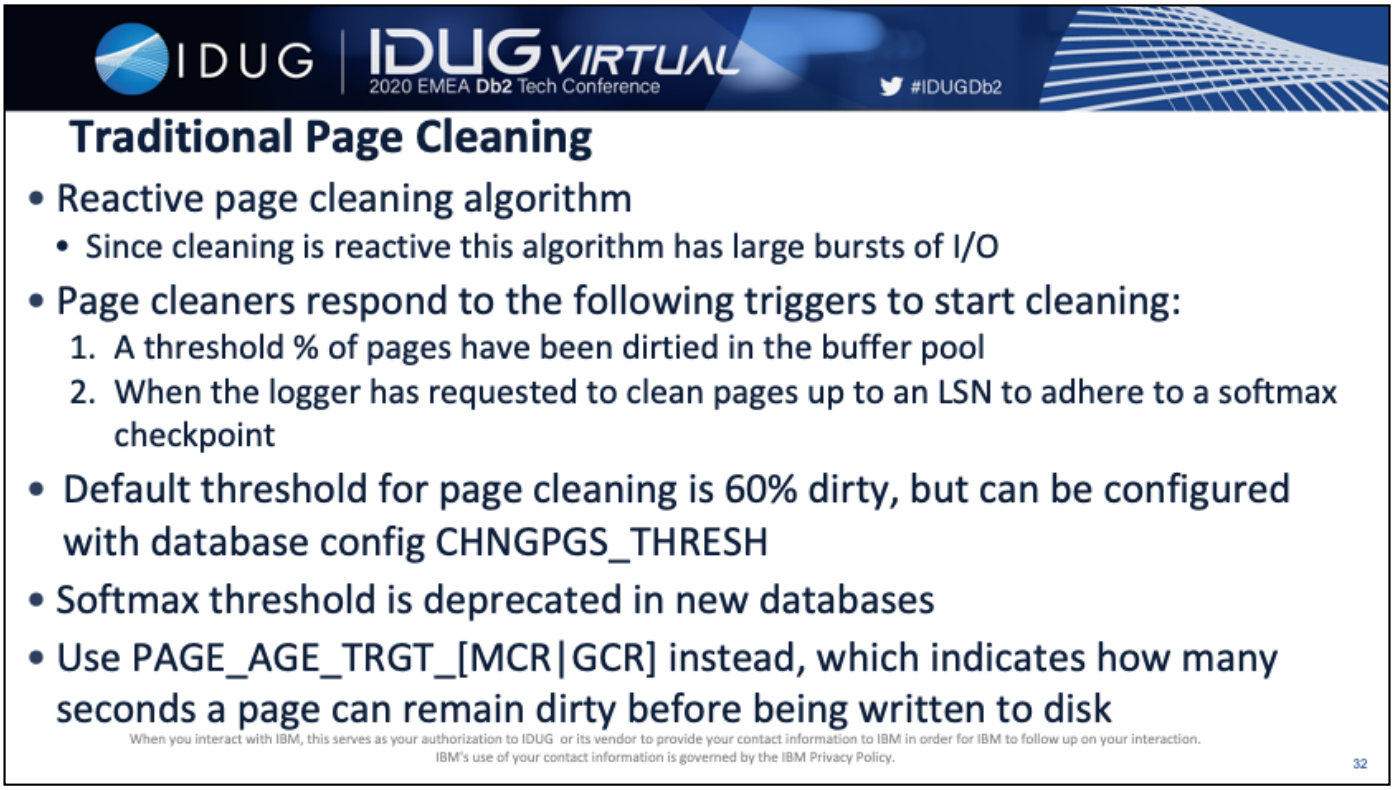
Page Cleaning

Page Cleaning Basics



- Page cleaners are background EDUs that are responsible for writing dirty pages to disk
- The number of page cleaners is configured with database config NUM_IOCLEANERS
- Writing dirty pages to disk using page cleaners offloads I/O from fixers, increasing overall database performance
- Ensuring that page changes are written to disk in a timely fashion helps to ensure a reasonable recovery time in the event of a crash
- There are 2 page cleaning algorithms

Default number of page cleaners is the same as the number of logical CPU cores, up to a max of 255

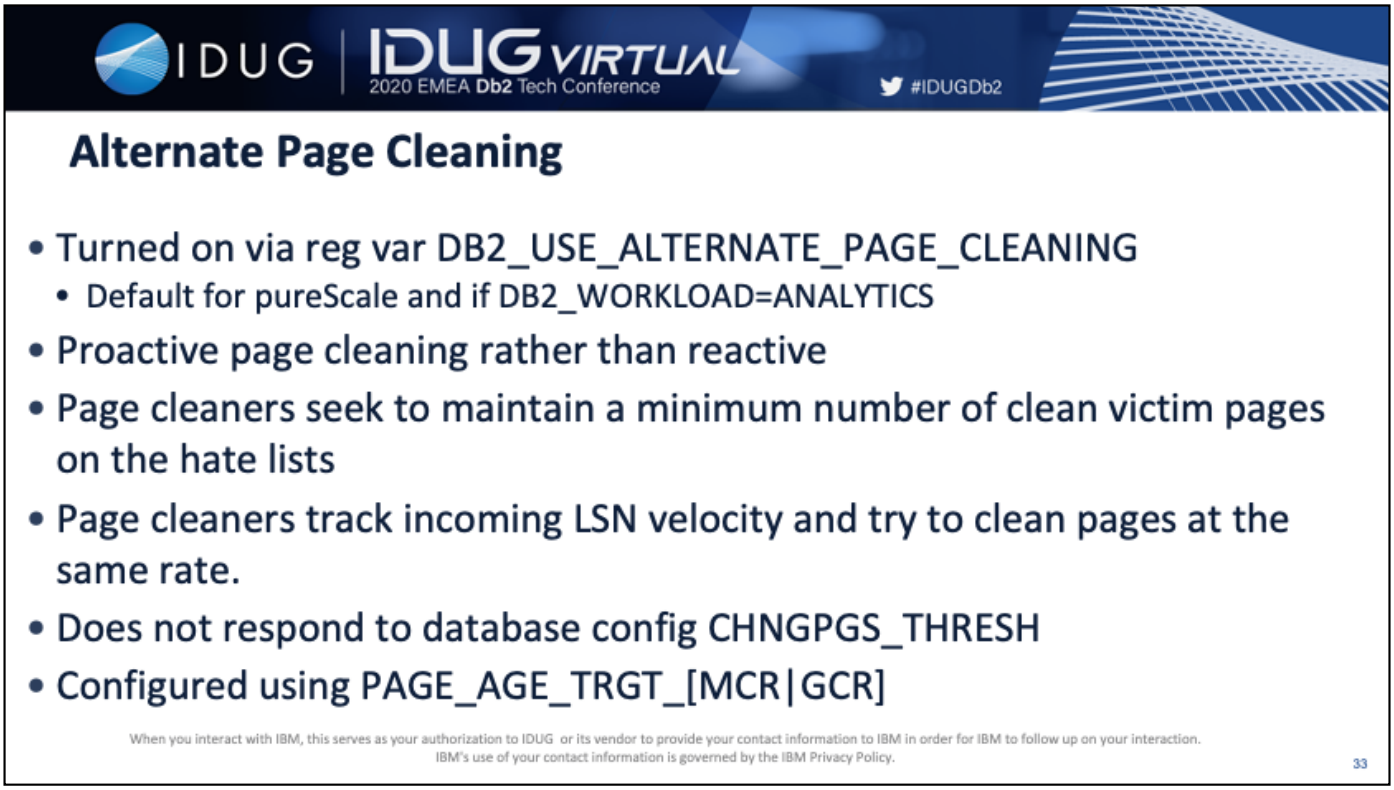


The softmax concept enables us to limit the number of log files that must be replayed during crash recovery due to dirty pages that have not been written out.

mcr = member crash recovery

The page_age_trgt_* settings are important in order to ensure pages that are dirty stay in the bufferpool for a number of seconds before being written out in case they need to be modified again.

gcr = group crash recovery



Pages are consistently cleaned such that bursts of I/O shouldn't happen

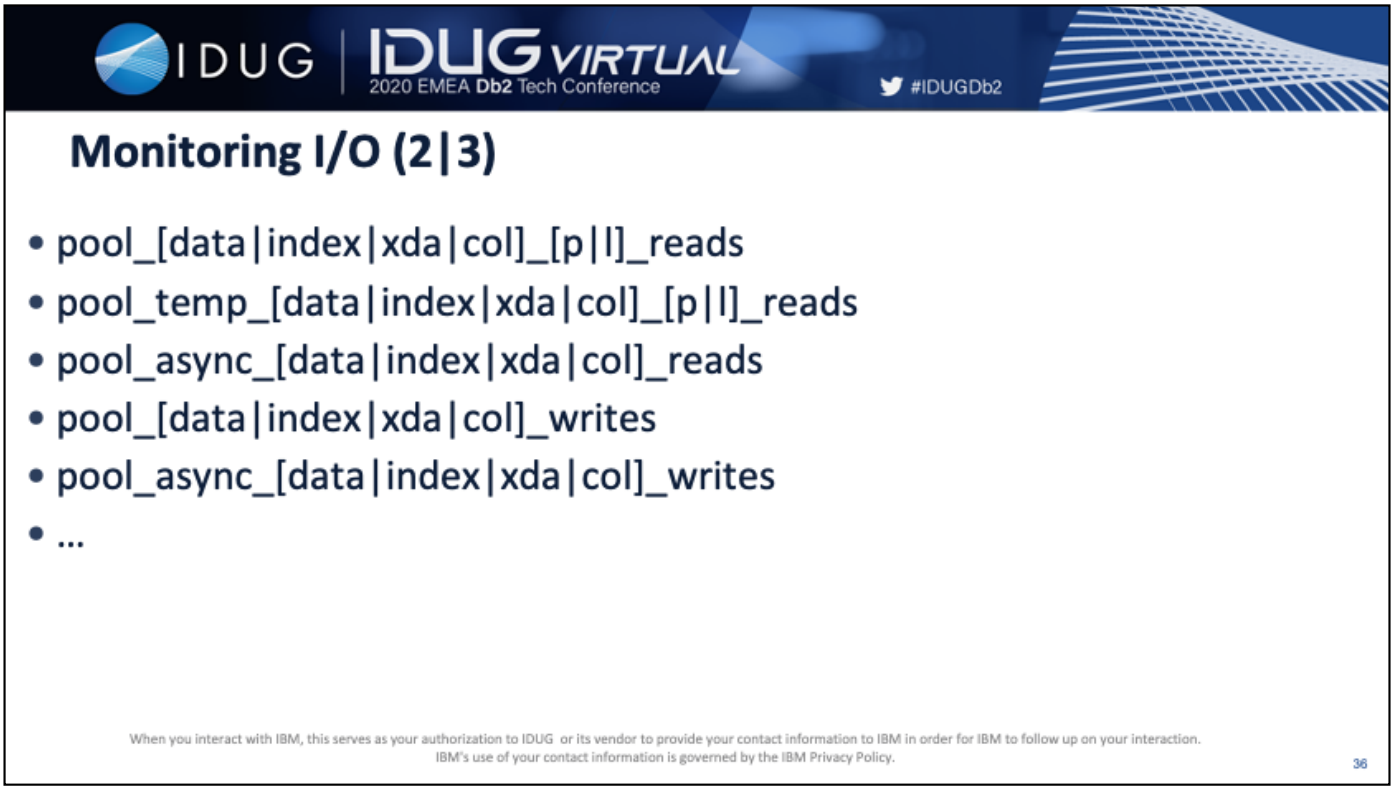
This type of page cleaning will often yield more stable workload performance, particularly for high-I/O workloads. However, while the average throughput rate for the workload may be higher with this type of page cleaning, some tests have shown that maximum throughput may be lower than with the traditional page cleaning algorithm.

There is a push to make APC the default page cleaning algorithm, but there is a possibility that doing so could have a slight negative affect on certain legacy workloads.

Some Key Tuning I/O Tips, Tricks, and Best Practices

Monitoring I/O (1|3)

- There are multiple ways to monitor I/O
- For the purposes of I/O analysis, we will focus on the table functions MON_GET_BUFFERPOOL, MON_GET_TABLESPACE, and MON_GET_CONTAINER since they are query-able
- Info on the various return values of those table functions can be found here:
  - https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.sql.rtn.doc/doc/r0053942.html
  - https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.sql.rtn.doc/doc/r0053943.html
  - https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.sql.rtn.doc/doc/r0053944.html



There are too many important monitor elements to list. Here are a few key ones to get started (granularity is pages):

- Physical/logical reads for data, index, XDA, and COL objects
- Physical/logical reads for data, index, XDA, and COL temporary objects
- Asynchronous physical reads for data, index, XDA, and COL objects
- Writes for data, index, XDA, and COL objects
- Asynchronous writes for data, index, XDA, and COL objects

LBP and GBP variants exist for many of these measures.

Monitoring I/O (3|3)

- Snapshot monitor support in various forms is deprecated in the latest versions of Db2
- You can also use db2pd -db <dbname> –bufferpools to get information about each buffer pool
- Info on the returned values is available here:
  https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0011729.html#r0011729__pdbufferpools

How Large Should my Buffer Pool(s) Be? (1|2)

- Typically, bigger is better to avoid physical I/O
- The size of a buffer pool is specified in terms of pages
  - Total size of the page area = <number of pages> * <PAGESIZE>
- Specify AUTOMATIC if you would like STMM to be able to resize the buffer pool
- Enabling AUTOMATIC buffer pool resizing via STMM depends on your required workload characteristics
  - Up to 10% extra memory is required during the resize simulation process
  - Resizing may impact workload performance
  - AUTO-resizing enables buffer pools to adapt to changing workload conditions without re-configuration

Db2 can handle large buffer pools efficiently

How Large Should my Buffer Pool(s) Be? (2|2)

- For recommendations on (initial) buffer pool sizing plus other configuration values, the AUTOCONFIGURE command may be executed
  - If the APPLY NONE clause is used, you are able to see recommendations before they are applied
  - https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0008960.html
- While sizing the buffer pool, keep in mind other memory requirements for your workload
  - E.g. Db2 BLU requires large amounts of sort heap memory for query processing and utility heap memory for dictionary management
  - AUTOCONFIGURE is able to take many of these requirements into account using the analytics_env option

AUTOCONFIGURE is a way to invoke the Db2 Configuration Manager manually

Should I Separate Objects Across Buffer Pools?

- There is no easy way to answer this question without understanding the workload(s)
- If there are multiple workloads that require different usage patterns and performance characteristics, it may be wise to use separate table spaces and buffer pools
  - Properly sizing multiple buffer pools may be challenging
- One excellent alternative is to leverage block-based buffer pools whenever possible
  - Block-based buffer pools effectively make a buffer pool multi-purpose to support both vectored and block I/O

Configuring Block-Based Buffer Pools (1|2)

- Reserving between 1 and 5% of the buffer pool for blocks may have a notable positive impact on workload performance
  - The NUMBLOCKPAGES option specifies the total number of pages in the block area
  - The BLOCKSIZE option specifies the number of pages in each block
- For optimal performance, table spaces with extent size = BLOCKSIZE should be bound to the buffer pool
- Once initially configured, it is worth examining monitor output from a running workload

Configuring Block-Based Buffer Pools (2|2)

- At the buffer pool, table space, and container level:
  - (pages_from_block_ios/block_ios) should be similar to buffer pool BLOCKSIZE
- If you know that your access pattern for the buffer pool for part of your workload should be entirely sequential:
  - (block_ios/(block_ios + vectored_ios) * 100) should approach 100
  - (pages_from_block_ios/(pages_from_block_ios + pages_from_vectored_ios) * 100) should approach 100
- If these guidelines do not hold, consider increasing the buffer pool's NUMBLOCKPAGES

Configuring I/O Parallelism

- The DB2_PARALLEL_IO reg var can be set to indicate if parallel I/O should be used for table spaces
  - This is particularly important if the table space's container(s) reside on more than one physical disk, as is the case for some RAID configurations
  - Usage is DB2_PARALLEL_IO=[TS ID]:[disks per container],…
  - * can be used as a wildcard for TS ID
  - Setting this reg var means that a table space's parallelism will be <num containers> * <disks per container>
- The table space EXTENTSIZE should be a multiple of the RAID stripe size

Configuring the Number of Prefetchers

- The NUM_IOSERVERS database config param should in most cases be set to AUTOMATIC
- Db2 does an excellent job selecting an appropriate number of prefetchers based on multiple factors including I/O parallelism and number of CPU cores if set to AUTOMATIC

Formula if num_ioservers=AUTOMATIC:

number of prefetchers = max(max(max over all table spaces(parallelism setting), number of cores * number of SMT threads), 16)

Configuring Table Space Prefetch Size

- The table space PREFETCHSIZE should either be AUTOMATIC (preferred) or a multiple of (the RAID stripe size * <disks per container>)
- Db2 does an excellent job selecting an appropriate PREFETCHSIZE based on multiple factors including number of containers if set to AUTOMATIC

Configuring Page Cleaners (1|2)

- Setting the NUM_IOCLEANERS database config param to AUTOMATIC is strongly recommended
- Setting reg var DB2_USE_ALTERNATE_PAGE_CLEANING=ON is also strongly recommended
- At the buffer pool and table space level:
  - ((pool_async_data_writes + pool_async_index_writes + pool_async_xda_writes + pool_async_col_writes)/(pool_data_writes + pool_index_writes + pool_xda_writes + pool_col_writes)) * 100 should be at or near 100
- If not, look closer at other page cleaner stats

Formula if num_iocleaners=AUTOMATIC:

number of page cleaners = max(ceil(# CPUs / # local logical database partitions) - 1, 1)

Configuring Page Cleaners (2|2)

- If APC=ON
  - The pool_no_victim_buffer element should be low relative to the number of logical reads
  - If not, consider increasing the NUM_IOCLEANERS database config param and/or decreasing the PAGE_AGE_TRGT_[MCR|GCR] database config params
- If APC=OFF
  - The pool_drty_pg_steal_clns element should ideally be as close to 0 as possible
  - Consider decreasing the CHNGPGS_THRESH database config param, if the softmax database config param is non-zero, set it to 0 and instead set the PAGE_AGE_TRGT_[MCR|GCR] database config params, or if the softmax database config param was already 0, decrease the PAGE_AGE_TRGT_[MCR|GCR] database config params

Other Notable Prefetch-Related Elements

- prefetch_waits and prefetch_wait_time
  - Ideally approaching 0
  - If large values, agents are waiting on prefetchers performing physical I/O
  - Consider increasing table space PREFETCHSIZE and/or db config param NUM_IOSERVERS
- unread_prefetch_pages
  - Ideally approaching 0
  - If large value, prefetched pages are being evicted before they are used
  - Would normally correlate with high synchronous physical reads
  - Consider increasing buffer pool size and/or decreasing table space PREFETCHSIZE

Buffer Pool Hit Ratio – The Key!

- Using the MON_GET_BUFFERPOOL table function, we can compute the overall buffer pool hit ratio as follows:
  - ((pool_data_lbp_pages_found + pool_index_lbp_pages_found + pool_xda_lbp_pages_found + pool_col_lbp_pages_found - pool_async_data_lbp_pages_found - pool_async_index_lbp_pages_found - pool_async_xda_lbp_pages_found - pool_async_col_lbp_pages_found) / (pool_data_l_reads + pool_index_l_reads + pool_xda_l_reads + pool_col_l_reads + pool_temp_data_l_reads + pool_temp_xda_l_reads + pool_temp_index_l_reads + pool_temp_col_l_reads)) * 100
- After all previous tuning is complete, if this value is still low, consider other techniques such as creating additional indexes or updating table space and buffer pool definitions based on the workload

If hit ratio is low, consider increasing buffer pool size, increasing prefetch size, adding prefetchers, or increasing size of block area. Also, consider workload characteristics to see if multiple buffer pools would provide benefit.

Chris is a Senior Software Engineer who is product owner for storage and compression for Db2 with BLU Acceleration. This technology is part of Db2 for Linux, UNIX and Windows, Db2 Warehouse, Db2 on Cloud, IBM Integrated Analytics System (IIAS) and Db2 Big SQL.