



DB2 for z/OS INSERT Performance

John Campbell

Distinguished Engineer

DB2 for z/OS Development



GUIDE Share France

Une Association Indépendante d'Utilisateurs IBM

Réunion du Guide DB2 pour z/OS France
Vendredi 21 novembre 2008
Tour Manhattan BMC, Paris-La Défense

Agenda

- Typical Performance Bottlenecks and Tuning
 - Read and Write I/O for Index and Data
 - Active Log Write
 - CPU Time
 - Lock/latch Contention and Service Task Waits
 - Use of Multi Row Insert
 - Use of INSERT vs. LOAD utility
- DB2 9 Performance Enhancements
 - Reduced LRSN Spin and Log Latch Contention
 - Increased Index Look aside
 - Asymmetric Index Leaf Page Split
 - Randomized Index Key
 - Table APPEND option
- Summary



Typical Performance Bottlenecks and Tuning Observations



Read and Write I/O for Index and Data

- Random Insert

- N synchronous read I/Os for each index
 - N depends on # index levels, # leaf pages, and buffer pool availability
 - Index read I/O time = $N * \text{\#indexes} * 2 \text{ ms}$
- Data read I/O time = 2 ms (0 if insert to the end)
- Deferred write I/O for each page
 - 2 ms for each row inserted
 - Depends on channel type, device type, I/O path utilisation, and distance between pages
- Keep the number of indexes to a minimum



Read and Write I/O for Index and Data ...

- Sequential Insert to the end of data set
 - For data, and ever-ascending or descending index key, insert
 - Can eliminate read I/O
 - Deferred write I/O only for contiguous pages
 - 0.4 ms per page filled with inserted rows
 - Time depends on channel type, device type and I/O path utilisation



Read and Write I/O for Index and Data ...

- Recommendations on deferred write thresholds
 - VDWQT = Vertical (dataset level) Deferred Write Threshold
 - Default: when 5% of buffers updated from one dataset, a deferred write is scheduled
 - DWQT = bufferpool level Deferred Write Threshold
 - Default: when 30% of buffers updated, a deferred write is scheduled
 - VDWQT=DWQT=90% maximum for bufferpool for data or index entirely resident (100% hit ratio) and repetitively updated Sequential Insert to the end of data set



Read and Write I/O for Index and Data ...

- With a high deferred write thresholds, write I/Os for data or index entirely resident in buffer pool can be eliminated except at system checkpoint or STOP TABLESPACE/DATABASE time
- Use VDWQT=0% for buffer pool for data with low hit ratio (1-5%) if single thread insert
 - Else $VDWQT = 150 + \text{\#concurrent threads inserting to this dataset}$
 - When 250 buffers are updated for this dataset, 128 LRU buffers are scheduled for write
- VDWQT=0% for sequential index insert
- Use default if not sure, also random index insert



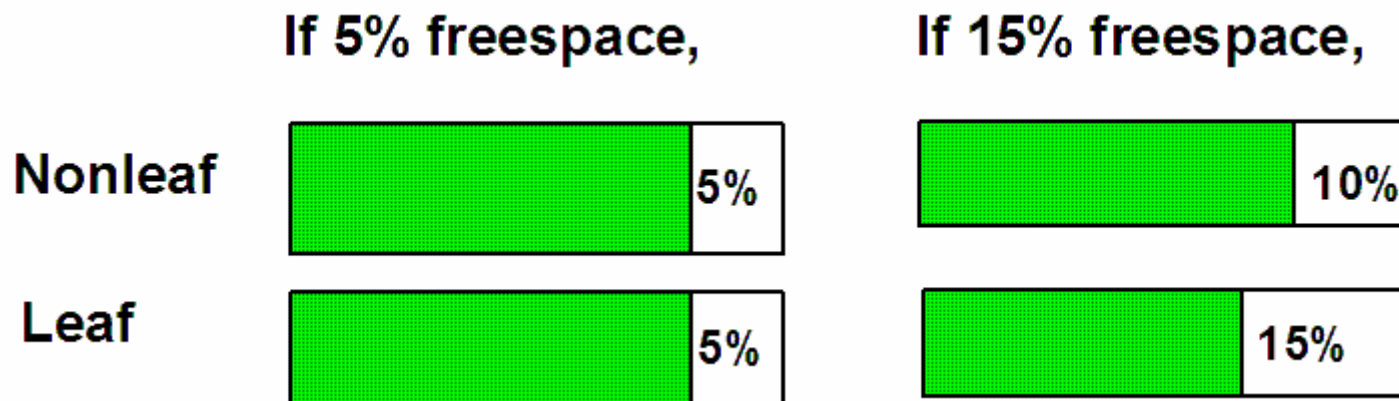
Freespace

- Use freespace
 - For efficient sequential read of index
 - For efficient sequential read of data via clustering index
 - To minimize index split
- Default freespace
 - 0 free page (FREEPAGE)
 - 5% freespace within data page (PCTFREE)
 - 10% freespace within index page (PCTFREE)



Freespace ...

- %freespace for non-leaf pages is limited to a minimum of 0 and maximum of 10%
 - So if 5% freespace for index, then 5% freespace for both leaf and non-leaf
 - If 15% freespace for index, then 15% freespace for leaf and 10% for non-leaf



Freespace ...

- For best insert performance
 - Random insert to index
 - Use index freespace
 - To reduce index leaf page splits
 - For efficient index sequential read
 - Use default %freespace unless you know better
 - Sequential insert to index
 - Immediately after LOAD, REORG, or CREATE/RECOVER/REBUILD INDEX
 - With 0% freespace, reducing the number of index pages and possibly index levels by populating each leaf page 100%
 - With 50% freespace, resulting in doubled index size
 - Use 0% freespace for data to reduce both read and write I/Os for each row insert
 - Possible penalty when reading multiple rows via clustering index



Freespace ...

- Trade off in free space search
 - Insert to the end
 - To minimize the cost of insert by minimising
 - Read/Write I/Os, Getpages, Lock requests
 - Search for available space near the optimal page
 - To store rows in clustering index sequence
 - To store leaf pages in index key sequence
 - To minimize dataset size
 - Search for available space anywhere within the allocated area
 - To minimise dataset size



Segmented Tablespace

- Segmented tablespace provides for more efficient search in fixed length compressed and true variable length row insert
 - Spacemap contains more information on available space so that only a data page with guaranteed available space is accessed
 - 2 bits per data page in non segmented tablespace ($2^{**}2=4$ different conditions)
 - 4 bits per data page in segmented tablespace ($2^{**}4=16$ different conditions)
- Also applies to Universal Tablespace (DB2 9)



MAXROWS n

- Optimisation to avoid wasteful space search on partitioned tablespace in fixed length compressed and true variable length row insert
- Must carefully estimate 'average' row size and how many 'average' size rows will fit comfortably in a single data page
- When MAXROWS n is reached the page is marked full
- But introduces on going maintenance challenges
 - Could waste space?
 - What happens if compression is removed?
 - What happens if switch from uncompressed to compressed?
 - What happens when new columns are added?



Partitioning

- Divide tablespace into partitions by key range
- Spread inserts over partitions
- Can reduce logical and physical contention to improve concurrency and reduce cost
- Separate index B-tree for each index partition of partitioned index (good for concurrency)
- Only one index B-tree for non- partitioned index (bad for concurrency)
- Over wide partitioning has potential to reduce number of index levels to reduce cost



Active Log Write

- Log data volume

- From DB2 log statistics, minimum MB/sec of writing to active log dataset can be calculated as

$$\frac{\text{\#CIs created in Active Log} * 0.004\text{MB}}{\text{-----}}$$

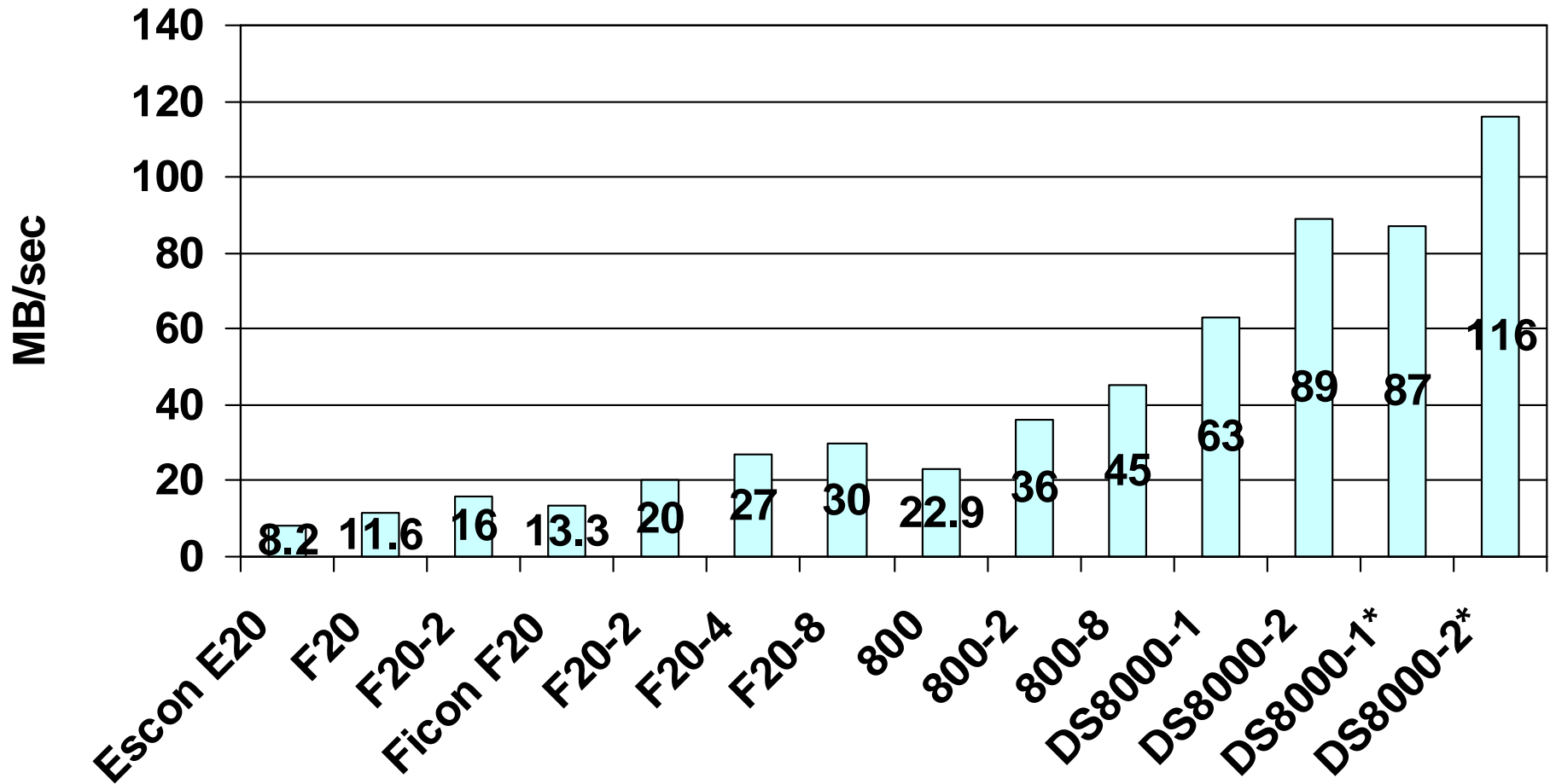
statistics interval in seconds

- Pay attention to log data volume if >10MB/sec
 - Optimise table design to minimise log record size
 - Consider use of DB2 data compression
 - Use faster device as needed
 - Consider use of DFSMS striping



Maximum Observed Rate of Active Log Write

- First 3 use Escon channel, the rest is Ficon
- -N indicates N I/O stripes; * MIDAW



Insert CPU Rough Rule of Thumb

	9672-Z17 CPU time
No index	40 to 80us
One index with no index read I/O	70 to 140us
One index with index read I/O	130 to 230us
Five indexes with index read I/O	500 to 800us

To get the CPU time for other processor models, see WWW.S390.IBM.COM/LSPR on Internal Throughput Ratios of various IBM processors



Insert CPU Rule of Thumb ...

- 9672-Z17 CPU time = 40 to 80us
 - + 30 to 50us * number of indexes
 - + 40us * number of I/O's
- Example
 - If 1 index and no read I/O because of sequential index insert, 40 to 80us + 30 to 50us = 70 to 130us
 - CPU cost for write I/O can be ignored because of sequential write of contiguous pages
 - If 3 indexes and 1 random read I/O for each index, 40 to 80us + (30 to 50us)*3 + 40us*3*2 (read +write) = 370 to 470us



Lock/Latch and Service Task Waits

- Rule-of-Thumb on LOCKSIZE
 - Page lock (LOCKSIZE PAGE|ANY) as design default and especially if sequentially inserting many rows/page
 - Could consider use of Row lock (LOCKSIZE ROW) if randomly inserting 1 row/page
- Page P-lock contention in data sharing environment
 - Data page update when LOCKSIZE ROW
 - Index page update
 - Spacemap page update



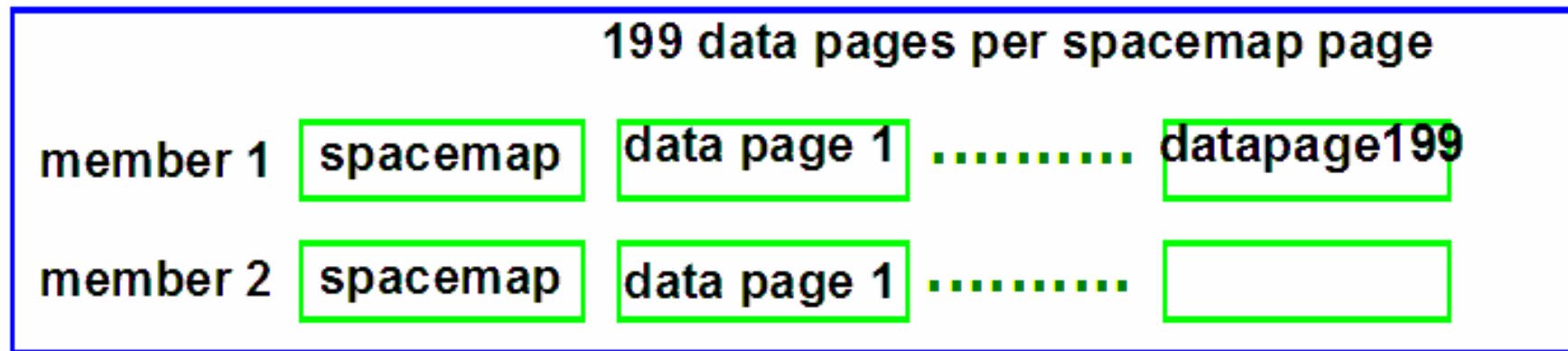
MEMBER CLUSTER

- Member-private spacemap and corresponding data pages
- Beneficial in data sharing environment to reduce page P-lock and page latch contention
 - Spacemap page
 - Data page if LOCKSIZE(ROW)



MEMBER CLUSTER ...

- Rows inserted by Insert SQL are not clustered by clustering index
- Instead, rows stored in available space in member-private area
- Rows inserted by Insert SQL are not clustered by clustering index
- Instead, rows stored in available space in member-private
- Option not available on segmented tablespace or UTS



TRACKMOD NO

- Reduces space map contention in data sharing environment
- DB2 does not track changed pages in the space map pages
- It uses the LRSN value in each page to determine whether a page has been changed since last copy
- Trade off as degraded performance for incremental image copy because of tablespace scan



DB2 Latch Contention in Heavy Insert Application

- Internal DB2 latch contention by Class 1 thru 32 shown in DB2PM Statistics Report Layout Long
- Index split in data sharing results in 2 forced physical log writes
 - Index split time can be significantly reduced by using faster active log device
- Reducing Class 19 log latch contention
 - Use LOAD LOG NO instead of SQL INSERT
 - Make sure Log Output Buffer fully backed up by real storage
 - Eliminate Unavailable Output Log Buffer condition
- Rule-of-Thumb on latch contention rate
 - Investigate if > 10000/sec
 - Ignore if < 1000/sec



DB2 Latch Contention in Heavy Insert Application

- Class 6 for latch for index tree P-lock due to index split
 - Data sharing only
 - Index splits in random insert can be reduced by providing non-zero %freespace
 - Also to minimise this, use fastest possible device, such as ESS, for active log
- Class 19 for logical log write latch
 - Both non-data sharing and data sharing
- If >1K-10K contentions/sec, disabling Accounting Class 3 trace helps to significantly reduced CPU time as well as elapsed time



Identity Column and Sequence Object

- DB2 to automatically generate a guaranteed-unique number for sequencing each row inserted into table
- Much better concurrency, throughput, and response time possible
 - Compared to application maintaining a sequence number in one row table, which forces a serialisation (one transaction at a time) from update to commit
 - Potential for 5 to 10 times higher insert/commit rate
- Option to cache (default of 20), saving DB2 Catalog update of maximum number for each insert
 - Eliminating GBP write and log write force for each insert in data sharing
- Recycling or wrapping of identity column and sequence value



Service Task Waits

- Service task waits most likely for preformatting
 - Shows up in Dataset Extend Wait in Accounting Class 3 Trace
 - Typically up to 1 second each time, but depends on allocation unit/size and device type
 - Anticipatory and asynchronous preformat in DB2 V7 significantly reduces wait time for preformat
 - Can be eliminated by LOAD/REORG with PREFORMAT option and high PRIQTY value
 - Do not use PREFORMAT on empty segmented tablespace or UTS with high PRIQTY



Multi Row Insert (MRI)

- INSERT INTO TABLE for N Rows Values (:hva1,:hva2,...)
- Up to 40% CPU time reduction by avoiding SQL API overhead for each INSERT call
 - % improvement lower if more indexes, more columns, and/or fewer rows inserted per call
- ATOMIC (default) is better from performance viewpoint as multiple SAVEPOINT log records can be avoided
- Implication for use in data sharing environment (LRSN spin)
- Dramatic reduction in network traffic and response time possible in distributed environment
 - By avoiding message send/receive for each row
 - Up to 8 times faster response time and 4 times CPU time reduction



Use of SQL INSERT vs. LOAD Utility

- LOAD utility advantage
 - Reduced CPU, I/O, and elapsed time
 - LOG NO option to avoid log write I/O and log latch contention problems
 - Optional statistics collection and image copy while loading
 - Easier operation
- SQL INSERT advantage
 - Higher data availability: page or possibly row, instead of tablespace or partition, lock
 - More efficient for smaller number of rows, e.g. <1000
 - Freespace exploitation



Use of SQL INSERT vs. LOAD Utility ...

- While INSERT looks for free space near by, LOAD RESUME adds to the end of data set
 - INSERT can result in better space utilization and clustering index key sequence but more expensive processing
- Online LOAD RESUME combines easier operation of LOAD with INSERT advantages



Use of SQL INSERT vs. LOAD Utility ...

- LOAD can be 5 to 10 times better than SQL INSERT in terms of CPU time, I/O time, and elapsed time.
 - However, SQL INSERT can have better concurrency
- Rule-of-Thumb: for pure performance, 1000 to 10000 rows is a 'typical' break-even point
 - LOAD if more than 10000 rows
 - SQL INSERT if less than 1000 rows
- Single LOAD job for the whole table vs. concurrent parallel partition LOAD jobs, one for each partition
 - Parallel partition LOAD jobs faster when there is only one index
 - Single LOAD job with SORTKEYS option generally faster when there are multiple indexes



DB2 9 Performance Enhancements



Reduced LRSN Spin and Log Latch Contention

- Less DB2 spin for TOD clock to generate unique LRSN for log stream
- No longer holds on to log output buffer latch (LC19) while spinning
- For data sharing
 - Potential for reduced LC19 Log latch contention
 - Potential for CPU time reduction especially when running on faster processor



Increased Index Look aside

- Prior to DB2 9, for clustering index only
- In DB2 9, now possible for additional indexes where `CLUSTERRATIO >= 80%`
- Potential for big reduction in the number of index getpages with substantial reduction in CPU time



Asymmetric Leaf Page Split

- Design point is to provide performance relief for classic sequential index key problem
- Asymmetric index page split will occur depending on an insert pattern when inserting in the middle of key range
 - Instead of previous 50-50 split prior to DB2 9
 - Up to 50% reduction in index split
- Asymmetric split information is tracked in the actual pages that are inserted into, so it is effective across multiple threads across DB2 members
- Prior to APAR PK62214, DB2 9 only remembered the last insert position and a counter
- APAR PK62214 introduces changes to the tracking and detection logic, and it should work much better for data sharing
- The new approach remembers an insert 'range' and tolerates entries being slightly out of order
- It may still not be effective for large key sizes (hundreds of bytes), or if entries come in very bad order (i.e., they do not look sequential)
- But for simple cases like 3, 2, 1, 6, 5, 4, 9, 8, 7, 12, 11, 10 ... DB2 will be able to determine that the inserted entries are ascending



Randomised Index Key

- Index contention can be a major problem and a limit for scalability
- This problem is more severe in data sharing because of index page P-lock contention
- A randomized index key can reduce lock contention
- `CREATE/ALTER INDEX ... column-name RANDOM,` instead of `ASC` or `DESC`
- Careful trade-off required between lock contention relief and additional getpages, read/write I/Os, and increased number of lock requests
- This type of index can provide dramatic improvement or degradation!
- Recommend making randomized indexes bufferpool resident



Table APPEND option

- New APPEND option is provided for INSERT
 - CREATE/ALTER TABLE ... APPEND YES
- Will reduce longer chain of spacemap page search as tablespace keeps getting bigger
- But will drive need for more frequent tablespace reorganization
- Degraded query performance until the reorganization is performed
- Alternative to consider prior to V9 for data sharing
 - MEMBER CLUSTER and PCTFREE=FREEPAGE=0
 - Will switch between append and insert mode
 - Success depends on deletes and inserts being spread across members



Summary



Summary

- Keep the number of indexes to a minimum
- Tune deferred write thresholds and free space
- Tune physical table design and/or use of data compression to minimise log record size
- Use faster channel, faster device, DFSMS striping for active log write
- Use MEMBER CLUSTER and TRACKMOD NO reduce spacemap and associated data page contention in data sharing
- Use Identity column or sequence with caching to automatically generate a unique key
- Important new DB2 9 new features and function

