2023 NEDb2UG Technical Conference

Db2 for z/OS availability, efficiency and application stability enhancements

Frances Villafuerte, IBM

June 2023

Agenda

- Db2 13 zPARM simplification
- Usability and Availability enhancements
  - PBG crossed partition search enhancement
  - Table alteration enhancement
- Application enhancements
- Question

zPARM simplification

- Db2 13 continue to eliminate unnecessary zPARMs
- zParm are removed in Db2 13 because they are deprecated, obsolete, or rarely utilized. The behaviors follow the default settings
- Carefully evaluate the default settings for your environment
- List of the removed ZPARM and its default value :

- **DDF_COMPATIBILITY=, (sets as null)**
- **HONOR_KEEPDICTIONARY=NO,**
- **IX_TB_PART_CONV_EXCLUDE=YES,**
- **PLANMGMTSCOPE=STATIC,**
- **REALSTORAGE_MANAGEMENT=AUTO,**
- **DSVCI=YES,**
- **EXTRAREQ=100,**
- **EXTRASRV=100,**
- **IMMEDWRI=NO,**
- **MAXARCH=10000,**
- **MAXTYPE1=0,**
- **OPT1ROWBLOCKSORT=DISABLE,**
- **PARA_EFF=50,**
- **RESYNC=2,**
- **SUBQ_MIDX=ENABLE,**
- **TRACSTR=NO,**

zPARMs are updated with new default setting

- The default values of these zPARMSs are updated in Db2 13 to reflect best practices or more typical usage.
- List of updated zPARMs and its new default setting

1. **DDF** - From NO to **AUTO**
2. **FTB_NON_UNIQUE_INDEX** - From NO to **YES**
3. **MAXSORT_IN_MEMORY** – From 1000 (KB) to **2000** (KB)
4. **SRTPOOL** - From 10000 (KB) to **20000** (KB)
5. **EDM_SKELETON_POOL** - From 51200 to **81920**
6. **EDMDBDC** - From 23400 to **40960**
7. **MAXCONQN** - From OFF to **ON**
8. **MAXCONQW** – From OFF to **ON**
9. **NUMLKTS** - From 2000 to **5000**
10. **NUMLKUS** - From 10000 to **20000**
11. **OUTBUFF** - From 4000 (KB) to **102400** (KB)
12. **SEGSIZE** – Starting **Db2 12**, value 0 = SEGSIZE **32**
13. **PAGESET_PAGENUM** - From ABSOLUTE to **RELATIVE**

zPARM has max value changes

- DSMAX – Specifies the maximum number data sets that can be open at one time
- Support more open datasets and double DSMAX limit to 400,000
  - Actual open dataset limit is determined by the amount of memory consumed per open dataset below 2Gb bar and amount of 31- bit private storage
  - Number of open data set capability depends on amount of the private 2GB storage for each environment
  - Db2 APARs PH09189 & PH27493 in V12 improved Db2 open dataset management when reaching to the max
  - In conjunction with z/OS 2.5 which reduces below the bar memory consumption per dataset by ~35%
  - Optionally move SWB above 2Gb bar
  - Exploitation of new z/OS API in V13
- The new highest possible value is changed from 200,000 to **400,000**

Availability Enhancements

PBG

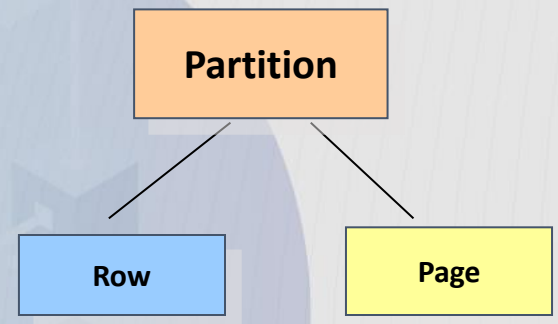
Availability & performance improvements in **Db2 13**

Lesson learned from the common use of PBG

- PBG table space can grow extremely large
  - A database administrator has a great challenge to tune the concurrent insert application for optimal performance
  - Unsuitable for XXL size large tables because complexity of performance and data management
- Some of common performance side effects from optimal performance design:
  - PBG conditional partition lock failure for multiple partitions cause application terminated prematurely
  - A randomly selected bi-directional cross-partitioned search to reduce contention results in longer space search for same cases
  - Performance side effects from searching full partitions
  - Application failed with incomprehensible message
    - ✓SQLCODE -904 with Reason code 00C90090 (Conditional lock failure) or 00C9009C (part is full and PBG exceed max partition defined)

PBG locking processing for crossed partition search

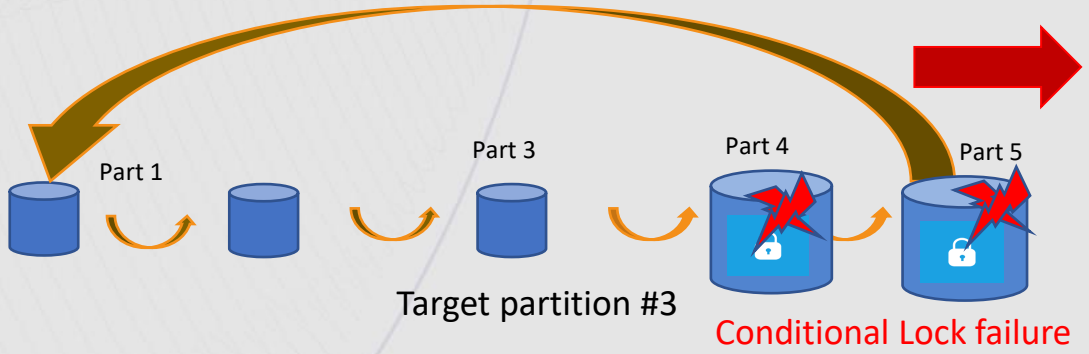
Universal Table Space Locking Hierarchy



- Follow locking hierarchy – partition lock is required to access the data
- Acquire partition lock conditionally
  - Conditional request of partition lock for optimal performance when PBG has multiple partitions
    - ✓ Unable to obtain conditional partition lock, the partition is skipped
  - After searching through all existing partitions once, Db2 fails INSERT transaction with -904 resource not available with reason code either 00C9009C(part is full and PBG exceed max partition defined) or 00C90090 (partition lock failure)
  - No new partition is added to avoid excess growth of the table space

For example

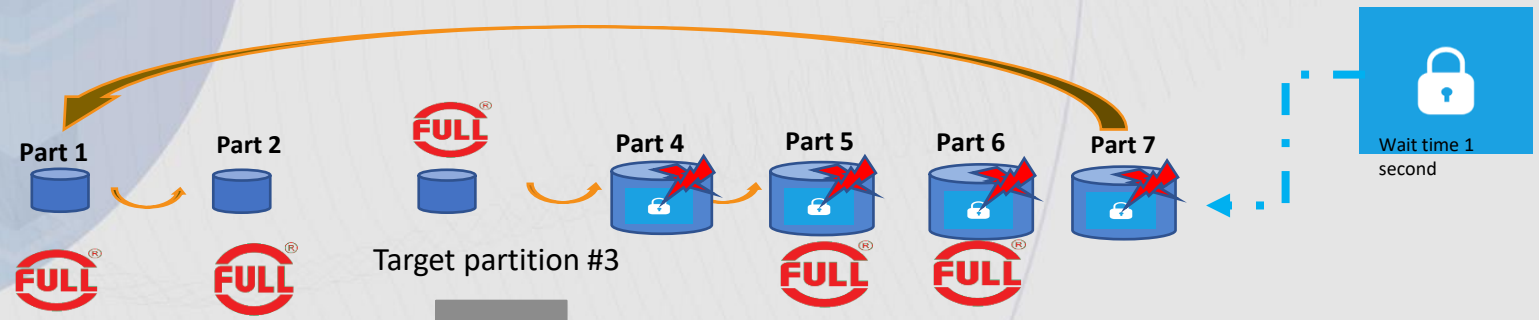
Searching partition in an ascending sequence by starting at partition 3 as determined by cluster index:
Assuming part 4 and part 5 are not full but failed with conditional lock
The partition searching sequence is 3->4->5->1->2 end all of searching



Insert failed and Issue SQLCODE -904 with reason code 00C9009C message

Achieving high availability with PBG in Db2 13

- Retry partition on partition failed conditional lock previously
  - Retry up to 5 non-full partitions
  - Selectively retry partition with conditional or unconditional partition lock
    - ✓ If retry unconditionally, limit lock wait time to be approximal 1-2 second
  - This is the last partition on the retry attempt array. Under this condition if the unconditional retry failed, then SQL -911 with 00C9008E message will be issued which can help the user to find out the
  - If all parts are retried and full, allowing adding a new partition
  - Provides higher insert successful rate and object availability



Retry:
retry part 4 and 7
because 5 and 6 are full

Part 4
Part 5
Part 6
Part 7

DSNT376I -DB2A PLAN=DSNTEP3 WITH 691
CORRELATION-ID=INSERTA4 CONNECTION-ID=BATCH
LUW-ID=DSNCAT.SYEC1DB2.DAE53CAF57D2=9
……

ONE HOLDER OF THE RESOURCE IS PLAN=DSNTEP3
WITH CORRELATION-ID=TQI01005 CONNECTION-ID=BATCH
…..
:*:* ON MEMBER DB2A
REQUESTER USING **TIMEOUT VALUE=1** FROM IRLMRWT
HOLDER USING TIMEOUT VALUE=60 FROM IRLMRWT

Db2 13 - IFCID 02 – Statistic Data

- New counters added to IFCID 02 Data Manager Data during the retry process

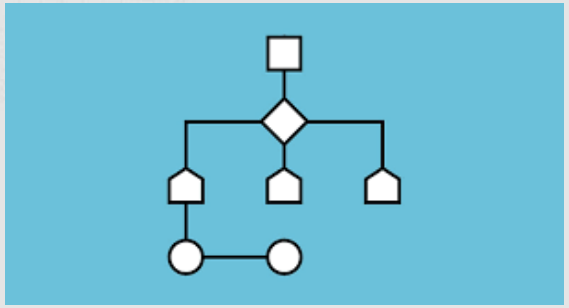
  - **CONDITIONAL LOCK FAILURE** (QISTCONDLKF)

    Number of failed conditional lock request during Insert operation

  - **UNCONDITIONAL LOCK RETRIES** (QISTRETRYLK)

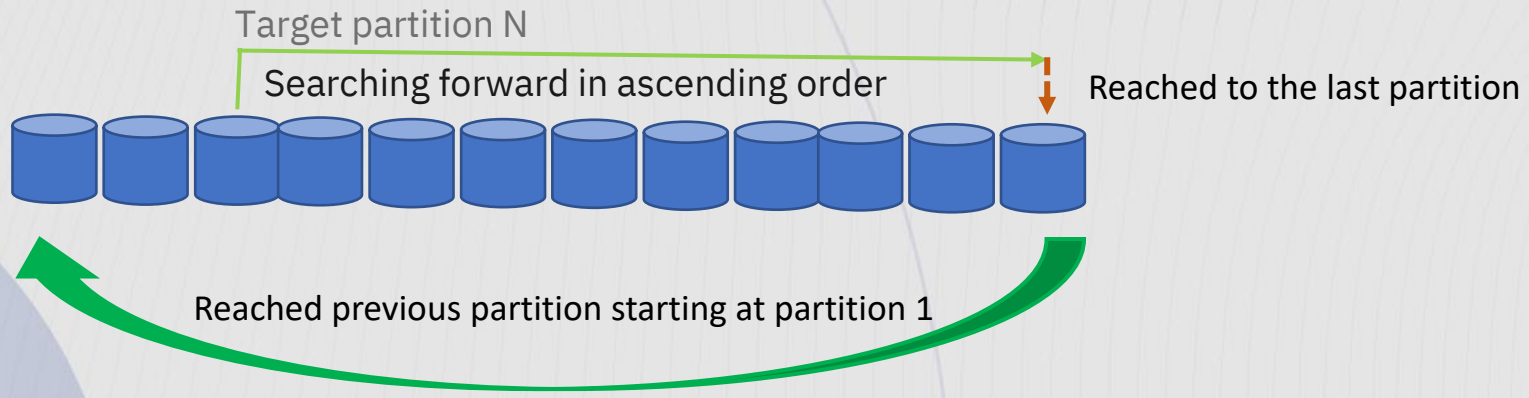
    Number of times a failed conditional lock request has been retried with an unconditional lock request

PBG
Crossed partition search algorithm



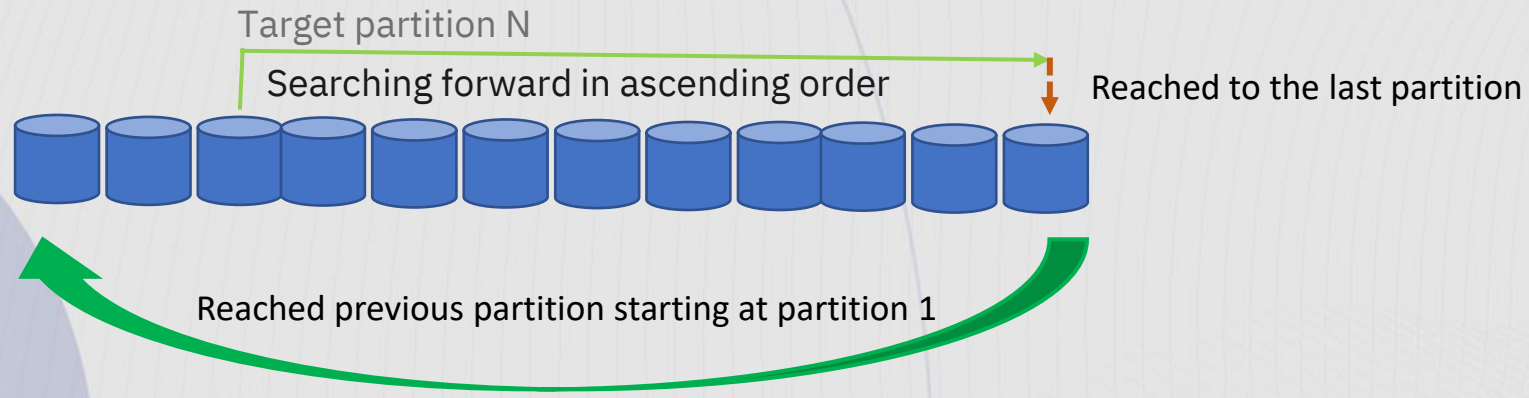
- For achieving optimal insert performance, Db2 12 delivered bi-directional crossed partition search algorithm for PBG table space
- A randomly selected bi-directional cross-partitioned search to reduce contention results in longer space search for same cases
- Enhancements from Db2 13

PBG crossed partition search algorithm – Ascending sequence



- Start with the target partition first, the one based on the clustering index key
- Searching forward in the ascending partition order
  - Starting at the target partition N, then N+1, N+2 ... Etc
  - When the last partition is full, then goes to the partition 1 and search forward to the target partition N
- Results in the accumulated small extra get page for each partition during the search
- Adds a new partition if no error from accessing each partition

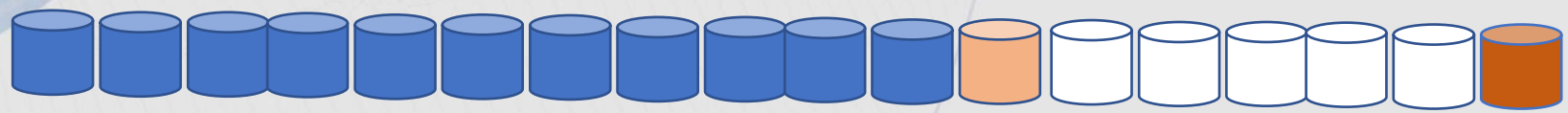
PBG crossed partition ascending search algorithm enhancements



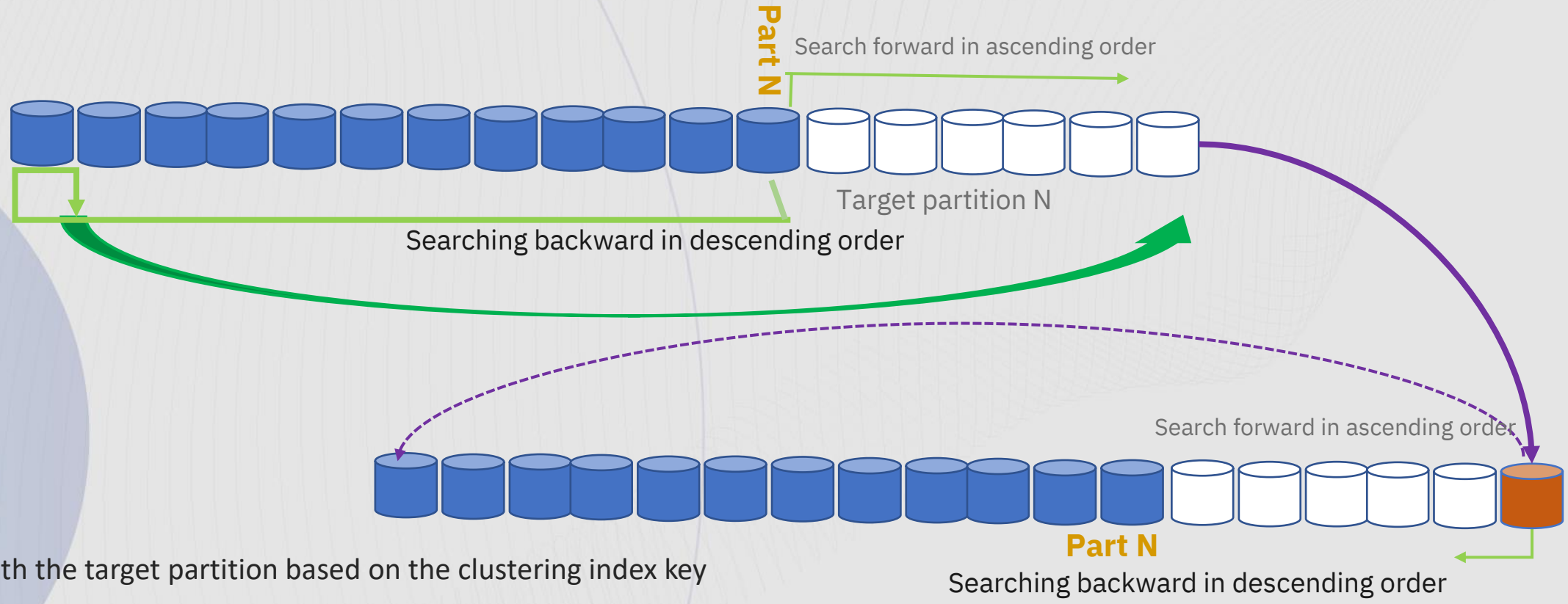
- Achieving higher performance with Db2 13
  - Optimization is in place to prevent every thread searching through full partitions
    - ✓ Full partition is tracked at each data sharing member
  - High % of performance improvement observed specially for random inserts
  - Best use case for object has more insert than delete
    - ✓ Small amount of the delete spread around the table space still requires searching of each partition
  - Workload balance among insert/delete operation is essential to take advantage of this improvement

Observation of descending PBG crossed partition search algorithm

- Embedded empty partitions between last defined partition and the last non-empty partition
  - Happen more often insert after REORG
- Empty partitions are harder to manage
- Possible more partition to search

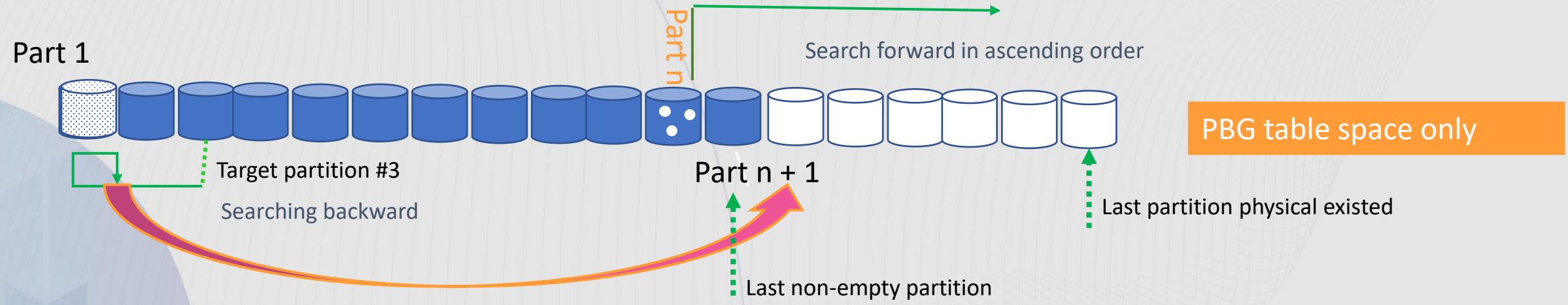


PBG crossed partition search algorithm – Descending sequence



- Start with the target partition based on the clustering index key
- Searching in a descending partition order
  - Starting at the target partition N, then N-1, N-2 ... Etc
  - When reaches to partition 1, then goes to the last partition of table space
- Side effects of descending crossed-partition search
  - It is designed in a way only beneficial for some niche use cases but creates side effects for others
  - Space is hard to manage
  - Embedded empty partition requires REORG on entire TS in order to reclaim the space

Improvement of PBG Descending crossed-partition Search



When descending partition search is used:

- Search in sequential descending order as before
- When partition 1 is reached, Utilizes RTS information to determine the next searching partition
  - Internally evaluates free space from RTS between last tracked non-empty partition and last partition
  - In doing so, partitions in between the original partition and the destination partition will be skipped
- Last non-empty partition is tracked after physical open through life of transactions
  - Possible still have embedded empty partitions between last defined partition and the last non-empty partition
  - For the case where Db2 does not have last non-empty partition tracked after physical open, and the target search starts with partition 1, the next target partition will be the last defined partition.

Usability Enhancements

Object conversion

Journey of Table Space conversion continue…

Simple Table Space or

Segmented table space

ALTER TABLESPACE …**MAXPARTITIONS** n

A single table in the table space

Partition by Growth

Classic Partitioned table space

ALTER TABLESPACE **…SEGSIZE n**

Db2 V12RM504

UTS
Partition by Range

Limit key on each partition

Segmented table space

Multiple Tables

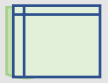
ALTER TABLESPACE …**MOVE TABLE** Tbname

TO TABLESPACE newtsname

Db2 12RM508

Partition by Growth

Partition by Growth



UTS Partition by Rang (PBR UTS)

ALTER TABLESPACE …**PAGENUM RELATIVE**

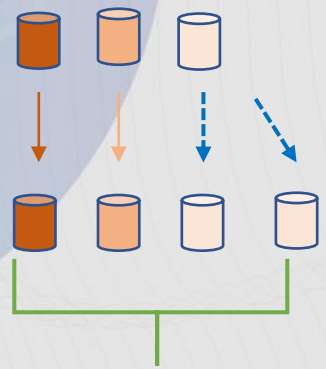
UTS PBR RPN

Lift size limitation

Object Conversion Evolution – PBG to PBR

- Achieving better availability with Online conversion from PBG to PBR in Db2 13
  - The next major step in online schema evolution strategy
  - Addresses increasing size of PBG table spaces that are better suited to PBR
  - Building on pending alter functionality with new ALTER syntax
  - Default to PBR with **RELATIVE PAGE NUMBERING** table space unless explicitly specified

Partition by Growth



UTS

Partition by Range RPN

```
ALTER TABLE SCR001.TB01 ALTER PARTITIONING TO
                        PARTITION BY RANGE (ACCT_NUM)
                       ( PARTITION 1 ENDING AT (199),
                        PARTITION 2 ENDING AT (299),
                        PARTITION 3 ENDING AT (399),
                        PARTITION 4 ENDING AT (MAXVALUE));
```

Converting PBG to PBR Considerations

- Index management:
  - All indexes defined on a table in partition-by-growth (PBG) are non-partitioned indexes (NPI).
  - The existing NPI indexes on the table are handled as part of the conversion. Db2 will not change any aspects or attributes of these indexes.
  - Partitioned indexes (PI) will not be created during this conversion process. If desired, they can be created after the conversion has been materialized.
- The table must not contain:
  - LOB column(s)
  - XML column(s)
- The high limit key for the last partition
  - Ascending Key - The last partition requires MAXVALUE
  - Descending Key – The last partition requires MINVALUE

When to consider converting PBG to PBR RPN (1 of 2)

- Things to evaluate when considering to convert PBG to PBR
  - **Insert or query performance degradation due to size of table space**
    - ✓ Performance degradation can occur for large table in PBG tables spaces, the size of the table space is often a major cause.
    - ✓ Db2 13 also introduces Insert enhancements for crossed partition search that can help with certain large PBG table space situation and insert pattern
    - ✓ Size of LOB table space associated with base table
      - PBR table space is more flexible to control the size of LOB table space
  - **Problems associated with very large non-partitioned indexes (NPI):**
    - ✓ Performance can suffer for index access when it must traverse the deep and wide NPI index trees that can result with many partitions

When to consider converting PBG to PBR RPN (2 0f 2)

- Things to evaluate when considering to convert PBG to PBR
  - **Difficulty completing REORG to maintain data clustering**
    - ✓ REORGs can be essential for applications that depend on the clustering sequence of the data being in good order. However, when a PBG table space grows too large and contains a massive number of partitions, it is often difficult to complete the REORG in the requested available time frame.
  - **Lack of parallelism features support for utilities**
    - ✓ Internal utility parallelism capabilities are not fully supported for PBG table spaces.
  - **Limited support for partition-level utility operations**
    - ✓ The partition-level utility operations available for tables in PBR spaces with relative page numbering can greatly simplify object maintenance
  - **Finding meaningful partition limit key**
    - ✓ V12 provided hidden ROWID column as partition limit key, however, it only supports in CREATE statement
    - ✓ An Aha DB24ZOS-I-1398 request to support alter add ROWID column

Application stability managements

Row Change Timestamp column (RCTC)

- Use case:
  - A way to identify every time a row is added or changed in a table
  - A timestamp column which is manage by Db2 internally instead of application
  - With a Row Change Timestamp Column, the row change timestamp column value is set to the timestamp corresponding to the time of the insert or update operation.

```
CREATE TABLE ORDERS
(ORDERNO SMALLINT,
SHIPPED_TO VARCHAR(36),
ORDER_DATE DATE, STATUS CHAR(1),
CHANGE_TS_Col NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW
CHANGE TIMESTAMP)
```

- As the example above:
  - When a row is inserted into the ORDERS table, the CHANGE_TS_Col column for the row is set to the timestamp of the insert operation.
  - When a row is updated in ORDERS table, the CHANGE_TS_Col column for the row is modified to reflect the timestamp of the update operation.

Default value of alter add RCTC column prior to V13

➢Problem Statement

- After a row change timestamp column is added to a table via ALTER TABLE add column, users might receive unexpected row change timestamp values for rows that existed before the column was added.
- In the absence of an initial default value, Db2 returns page's last-modified timestamp instead. As a result, insert, delete, or update into a page will change the derived RCTC value of all other existing rows within the page. This produces unpredictable row change timestamp results, due to insert/update/delete to other rows on the same page.

```
CREATE TABLE TestTB1 (C1 INTEGER NOT NULL);

INSERT INTO TestTB1 VALUES (1);
INSERT INTO TestTB1 VALUES (2);

ALTER TABLE TestTB1 ADD COLUMN C2 NOT NULL GENERATED ALWAYS FOR
EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP;

INSERT INTO TestTB1 VALUES (3);

SELECT TestTB1.C2 FROM TestTB1 WHERE TestTB1.C1 = 1; ← Result is the
insert time of row C1=3
```

V13 better Default value alter add RCTC column

Solution in V13R1M503 or higher

- Use the ALTER TABLE timestamp as the default value in the absence of an initial default value in application compatibility level V13R1M503 or higher.
- Save this timestamp in the SYSIBM.SYSCOLUMNS.DEFAULTVALUE column
- Existing added RCTCs previously continue behaving as before: DEFAULTVALUE remains blank and Db2 uses the last-modified timestamp in the row's page header.
- The ROW CHANGE expression is changed to recognize this default value. Both the TIMESTAMP and TOKEN options return a constant timestamp and derived token for existing rows.

```
CREATE TABLE TestTB1 (C1 INTEGER NOT NULL);

INSERT INTO TestTB1 VALUES (1);
INSERT INTO TestTB1 VALUES (2);

ALTER TABLE TestTB1 ADD COLUMN C2 NOT NULL GENERATED ALWAYS
FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP;

INSERT INTO TestTB1 VALUES (3);

SELECT TestTB1.C2 FROM TestTB1 WHERE TestTB1.C1 = 1; ←
Returns as column of RCTC is added
```

Usability Enhancements

Application Locking Control

Application Locking control observation

- Lock timeout control problem zPARM (**IRLMRWT**)
  - The lock timeout interval in Db2 is governed by zparm IRLMRWT.
  - This is one-size-fits-all approach for all application in the same sub-system
  - Certain business critical application may be suitable for a longer lock wait time interval
  - It is for DBA to manage the zPARM setting to accommodate the multi-tenancy cloud environment with disparate application needs.
  - Online changeable after **V13R1M500** or higher

- DDL's specific subsystem parameter (**zPARMS DDLTOX**)
  - Data definition time out field – specially for DDL operation
  - Setting the factor of IRLMRWT before DDL timeout

- Lack of control on victim of deadlock
  - Certain DDL activities can be prone to deadlocks. If the thread performing the DDL is chosen as the victim then scheduled DDL activities may fail,
  - Requiring repeat attempts and/or impacting subsequent work.
  - Even DDLTOX can not influence Db2 selects the victim of deadlock

Common lock timeout scenario

- DDL encounters lock contention with RTS externalization process
  - Depends on the contention, it could result in timeout or deadlock situation
- DDL break-in issues
  - DDL change must complete, but it is deadlock with application (bind or rebind of the application)
- Application contention
  - Difficult to manage locking serialization among applications
  - Different applications have different characteristics but have no way to express their own priorities and wait time in case of lock contention.
  - Possible application by-pass may be to split off into its own Db2 subsystem to provide for application affinities or redesign application. However, this increases management cost and overhead, and decreases availability and resiliency
  - The application must be split off into its own Db2 system thereby creating application affinities causing increased management cost and overhead and decreased availability and resiliency

Application with lock timeout control

- Special Register - **CURRENT LOCK TIMEOUT** special register
  - Set current lock time out at application level through new special register
  - An application architect determines the application characteristics require either a longer or shorter lock timeout interval in second
  - Application compatibility level **V13R1M500** or higher.
  - It overrides the IRLMRWT subsystem parameter and controls the number of seconds to wait before a resource timeout is detected
  - The data type of the register is INTEGER, and the valid value is between -1 and 32767, inclusive, or the null value
  - The special register is also applicable to claims, drains and other waiting periods similar as locks, however, it is not applicable to certain processes such as P-locks or plan/package allocation (package locks)
  - The special register is also supported in profile tables for remote applications
  - New **SPREG_LOCK_TIMEOUT_MAX** zPARM parameter
    - ✓Controls the maximum value that can be specified in a SET CURRENT LOCK TIMEOUT statement
  - Setting the value with SQL statement

```
SET CURRENT LOCK TIMEOUT = 15 ( SQL statement)
```

CURRENT LOCK TIMEOUT special register Value

- **NULL**
  - This is the default value. In this case, the current value of the IRLMRWT subsystem parameter is used when waiting for a lock and when referencing the special register in SQL statements.
- **WAIT (-1)**:
  - A value of -1 specifies that timeouts are not to take place, and that the application is to wait until the lock is released or a deadlock is detected.
- **NOT WAIT (0)**:
  - A value of 0 specifies that the application is not to wait for a lock; if a lock cannot be obtained, an error is to be returned immediately.
- **WAIT integer or integer:**
  - Depending on the integer value, it either indicates the real timeout seconds or has the special meanings of -1/0 as described above.
- **Variable:**
  - Depending on the value contained in the variable and/or indicator variable, it has the same semantics as described above.

Application with deadlock control

- A new global variable - **SYSIBMADM.DEADLOCK_RESOLUTION_PRIORITY**
  - Influence deadlock resolution at application level through new global variable
  - Influence deadlock victim decision by external application
  - This built-in global variable specifies a relative priority to be used in resolving deadlocks with other threads
    - ✓The data type of the built-in global variable is SMALLINT, and the valid value is between 0 and 255, inclusive, or the null value
    - ✓The higher value of the global variable, the higher priority to get the resource
  - The global variable is also supported in profile tables for remote applications
  - There is no catalog change, however, DEADLOCK_RESOLUTION_PRIORITY built-in global variable is created through a CATMAINT job (V13R1M501).
  - Application compatibility level **V13R1M501** or higher

Improvements in trace records

- some of IFCID improvements
  - IFCID437 is a new trace record for execution of the SET CURRENT LOCK TIMEOUT statement
  - IFCID 2 and 3**:** new counters are introduced for accounting and statistics traces
  - IFCID 106: is modified to trace the new SPREG_LOCK_TIMEOUT_MAX subsystem parameter
  - IFCID 172: A new flag is added to indicate the deadlock value comes from global variable or the Db2 internal factors.

Better Application usability with improved message (1 of 2)

- DSNT376I message

DSNT376I PLAN=plan-name1 WITH CORRELATION-ID=correlation-id1 CONNECTION-ID=connection-id1 LUW-ID=luw-id1 THREAD-INFO=thread- information1 IS **TIMED OUT**.

ONE HOLDER OF THE RESOURCE IS

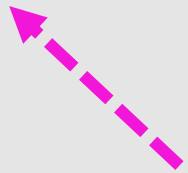
PLAN=plan-name2 WITH CORRELATION-ID=correlation-id2 CONNECTION-ID=connection-id2 LUW-ID=luw-id2 THREAD-INFO= thread-information2 ON MEMBER member-name.

REQUESTER USING TIMEOUT VALUE <tout-interval1> FROM <timeout-source1>.

HOLDER USING TIMEOUT VALUE <tout-interval2> from <timeout-source2>.

- ***tout-interval1*** and ***tout-interval2***

wait interval in seconds before a lock request timed out



- ***timeout-source1*** and ***timeout-source2***

the timeout interval specification that was used for the timeout, such as IRLMRWT or special register

Better Application usability with improved message (1 of 2)

- Example of DSNT376I message

```
DSNT376I -DB2A PLAN=DSNTEP3 WITH 691
   CORRELATION-ID=INSERTA4 CONNECTION-ID=BATCH
   LUW-ID=DSNCAT.SYEC1DB2.DAE53CAF57D2=9
   ……

   ONE HOLDER OF THE RESOURCE IS PLAN=DSNTEP3 WITH CORRELATION-ID=TQI01005 CONNECTION-ID=BATCH
   …..
   :*:* ON MEMBER DB2A
   REQUESTER USING TIMEOUT VALUE=10 FROM Special Register
   HOLDER USING TIMEOUT VALUE=60 FROM IRLMRWT
```

Summary

- zPARM simplification
  - Carefully evaluate delete and default changes zPARMs
- PBG improvements
  - Improvement of insert transaction successful rate with partition retry capability
  - Improved of crossed partition search algorithm to enhance availability and useability
  - Improved performance with smart way of track free space
- Object conversion case
  - Allow to convert PBG -> PBR RPN
  - Evaluates the need with adequate planning for the conversion
- Application management control
  - Improvement in adding the RCTC column
  - Lock timeout control at application level with special register - CURRENT LOCK TIMEOUT
  - DeadLock victim control at application level with global variable - DEADLOCK_RESOLUTION_PRIORITY
  - Profile supports both – new special register and global variable

Thank You

Speaker: Frances Villafuerte

Company: IBM

Email Address: Francesv@us.ibm.com