IDUG
The Worldwide DB2 User Community

IDUG DB2 Tech Conference
Denver, Colorado USA | May 2012

Deep Dive Into Storage Optimization When And How To Use Adaptive Compression

Thomas Fanghaenel
IBM

Session Code: H06
May 16, 2012 9:15am - 10:15am | Platform: LUW

Agenda

- Recap: Compression in DB2 9 for Linux, Unix and Windows
- New in DB2 10 for Linux, Unix and Windows
  - Adaptive row compression
  - Simplified row compression estimation
  - Log archive compression
- Best Practices
- Performance

Recap: Deep Compression in DB2 9

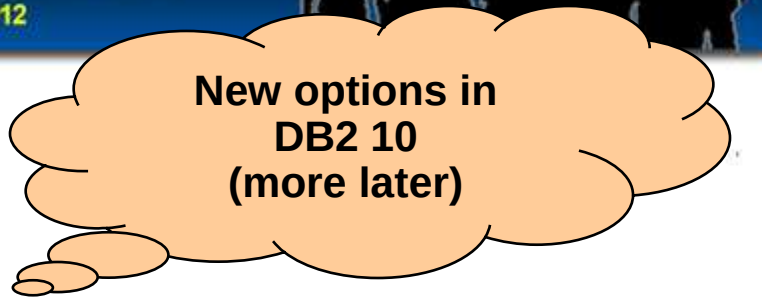
Value Compression

```
CREATE TABLE … VALUE COMPRESSION
ALTER TABLE … ACTIVATE VALUE COMPRESSION
```

- Uses alternative internal row format
- Slightly different storage space requirements than default row format
  - Less overhead for variable-length columns
  - More overhead for fixed-length columns
  - No storage overhead for NULL values and empty character strings
- Allows for storing system defaults with minimal overhead
  - For data types other than DATE, TIME, TIMESTAMP
  - Columns must have COMPRESS SYSTEM DEFAULT enabled

Row Compression

CREATE TABLE … COMPRESS YES
ALTER TABLE … COMPRESS YES



- Uses a dictionary-based compression algorithm
  - One compression dictionary per (data or range) partition
  - Dictionary contains recurring patterns found across entire partition
- Dictionary is created automatically (starting in DB2 v9.5) when row compression is enabled and partition grows beyond a threshold
- Dictionary can be refreshed via Classic Table Reorganization
  - Improve compression ratio when data patterns change over time

XML Compression (DB2 LUW v9.7)

- XML data is compressed when table is enabled for compression
  - No longer need to inline documents for compression purposes
- Uses similar technology as row compression, applied to the XML storage object
- Enabled automatically, whenever row compression is enabled
  - Requires XML storage object in DB2 LUW v9.7 storage format

Index Compression (DB2 LUW v9.7)

```
CREATE INDEX … COMPRESS YES
ALTER INDEX … COMPRESS YES
```

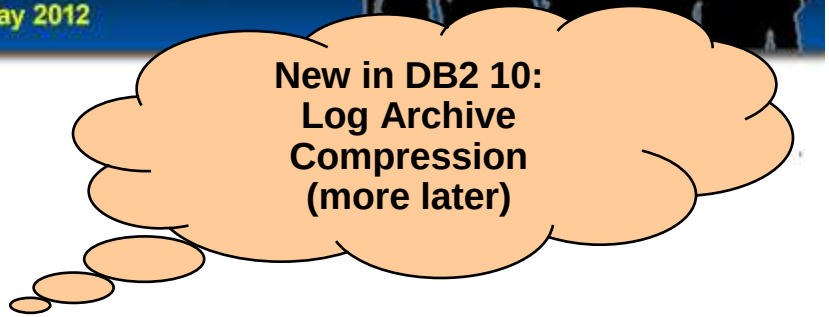
- Indexes on tables with row compression are compressed automatically
  - But: **`ALTER TABLE … COMPRESS YES`** doesn't apply this to existing indexes
- Uses three different compression techniques on leaf pages
  - Meta-data compression
  - RID-list compression
  - Key prefix compression
- Once enabled, index compression is dynamic and on-the-fly
  - No need to reorganize indexes to retain high compression ratios

Compression for Temporary Tables (DB2 LUW v9.7)

- Temporary tables and their indexes are automatically compressed
  - For declared global and system temporary tables
  - Uses automatic dictionary creation (ADC) with high threshold
  - Reqires license for Storage Optimization feature

Backup Compression

BACKUP DATABASE … COMPRESS



- Using row, index and XML compression reduces the size of the database
  - Backup images automatically get smaller
  - Backup time reduced
- Backup compression shrinks backup images to smallest possible size
  - Compresses metadata and LONG/LOB data, too
  - Can be a CPU-intensive operation
- Consider separation of data/index/XML/LOB into different tablespaces
  - Use tablespace-level backup to exercise finer control over compression settings

New in DB2 10: Adaptive Row Compression

Row Compression in DB2 10 – Terminology

Row compression means compressing each data row individually

In DB2 10 there are two different flavors of row compression:

Classic row compression

- Compress rows by means of a static partition-level compression dictionary
- Same row compression mechanism as in DB2 9

Adaptive row compression

- Compress rows by means of two compression dictionaries
  - Static partition-level dictionary
  - Dynamic page-level dictionaries
- New row compression mechanism in DB2 10

Adaptive Row Compression – Why?

- Classic row compression works very well in many cases
  - Very fast and robust
  - Not sensitive to data clustering and ordering
- Classic row compression has limitations
  - Dictionary requires classic table reorganization to refresh
  - Not sensitive to data clustering and ordering
  - Dictionary capacity limits compression ratios for some large tables
  - Theoretical maximum: 10x compression (90% savings)
- Different characteristics than competitors' compression

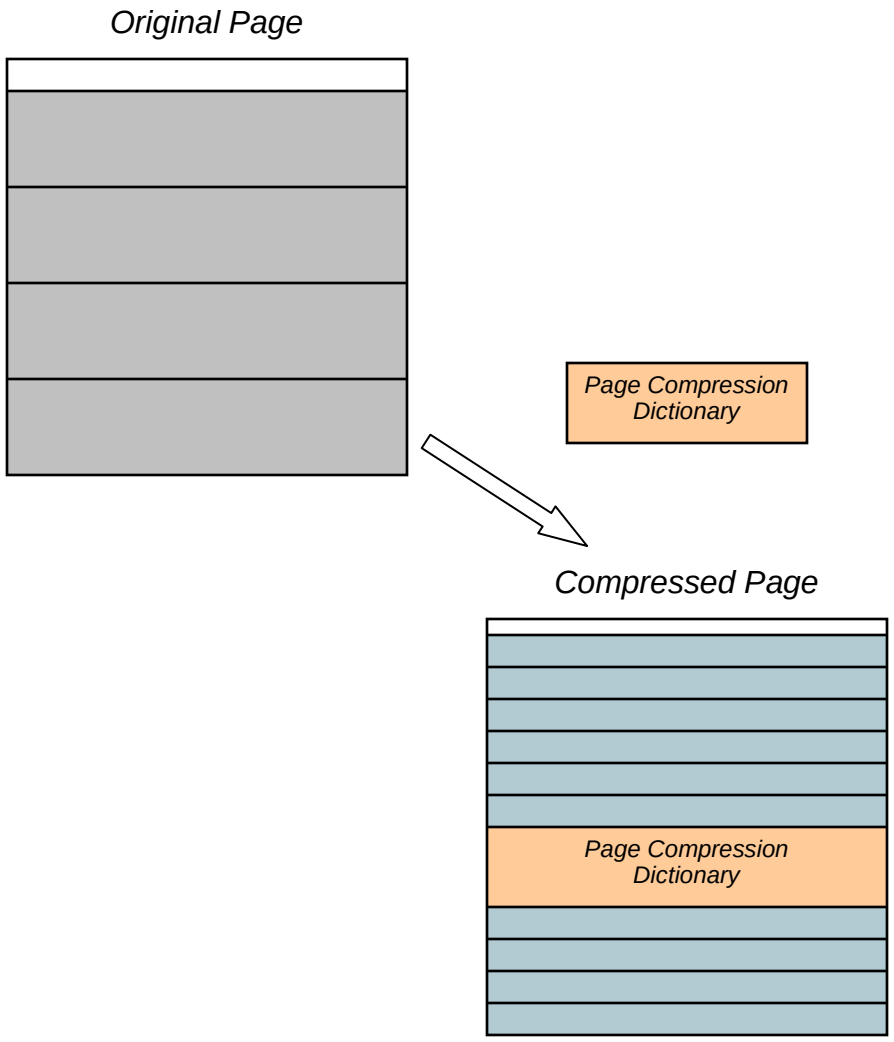
Adaptive Row Compression – Value Proposition

- Better compression
  - Classic row compression: Typically saves ~40%-75%
  - Adaptive row compression: Typically saves ~75%-85%
  - Adaptive typically saves 30% over classic row compression
- More automatic
  - Automatic compression in DB2 9 can produce up to 2x the data size of best compression (after REORG)
  - In DB2 10, storage requirements for automatic and best compression typically differ by less than 20%
- Reduced TCO
- Industry-leading compression technology for row stores

Adaptive Row Compression – Overview

- Combination of two compression algorithms
- Both algorithms detect different kinds of patterns
  - Globally recurring byte sequences
  - Locally recurring byte sequences
- Compression is applied for each row individually
  - Compression via table-level dictionary is applied first
  - When records are placed in a page, compression via page-level dictionary is applied
  - Rows are compressed during DML operations and admin tasks
    - INSERT, UPDATE, IMPORT, LOAD
    - REORG, REDISTRIBUTE

Example: Creation of Page-Level Dictionaries



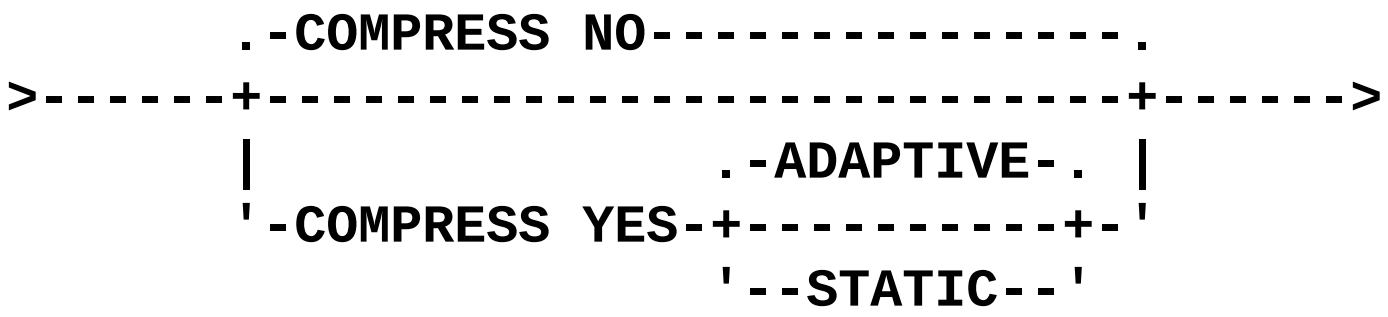
- Rows are inserted into a page
- When page is almost full, page gets compressed

1. Detect common recurring patterns in original records & build dictionary structure
2. Build compressed page by compressing all existing records
3. Insert page compression dictionary (special record)
4. Insert more compressed records in additional free space

Page Dictionary Management

- Management of page-level dictionaries is fully automatic
  - Creation happens when page is almost full
  - Recreation may be triggered when page becomes full again
    - Recreation is only triggered if actual compression ratio is significantly less than projected compression ratio (based on last dictionary build)
  - Deletion happens if record is placed in a committed empty page
- Database manager can decide to skip (re-)building page-level dictionaries
  - Decision is based on runtime stats (generated savings)
  - For dictionary recreation, age of existing dictionary is also considered
- Goal: No noticeable CPU overhead if adaptive compression generates little savings

Adaptive Row Compression – DDL Syntax



- Default for new tables in DB2 10 is adaptive compression
- Compressed tables keep classic row compression mode during upgrade to DB2 10

Simplified in DB2 10: Compression Estimation

Compression Estimation Functions in DB2 10

ADMIN_GET_TAB_COMPRESS_INFO_V97

Two execution modes in v9.7: 'REPORT' and 'ESTIMATE'.

Deprecated in Galileo, will be succeeded by two separate functions.

ADMIN_GET_TAB_COMPRESS_INFO

- One-stop-shop for estimation of compression savings
- "What could I save if I ALTER my table, and REORG now?"
- Displays all information compression savings for:
  - Current savings (accurate w/o Runstats)
  - Potential savings if table was
    - COMPRESS YES STATIC
    - COMPRESS YES ADAPTIVE
- Only one invocation necessary

ADMIN_GET_TAB_DICTIONARY_INFO

- Replaces 'REPORT' mode
- Result set matches the one of ADMIN_GET_TAB_COMPRESS_INFO in v9.7

Compression Estimation Functions in DB2 10 Example

- Invoke admin function:

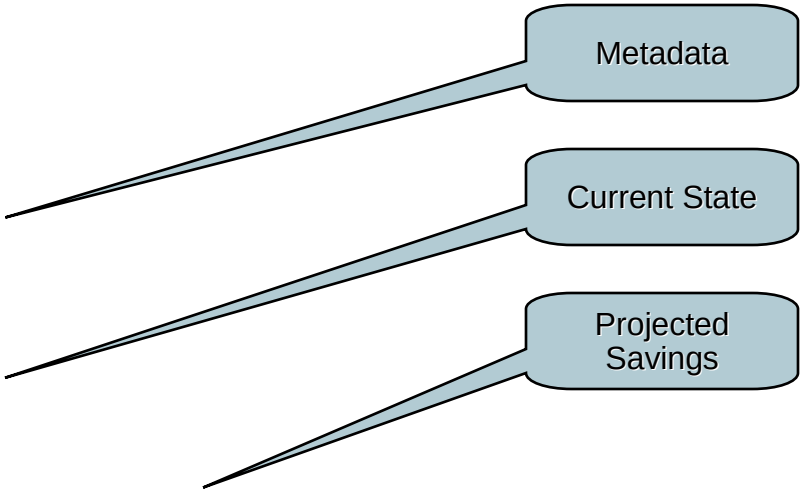
```
db2 => SELECT TABSCHEMA, TABNAME, DBPARTITIONNUM, ROWCOMPMODE, DATAPARTITIONID, OBJECT_TYPE
              PCTPAGESSAVED_CURRENT, AVGROWSIZE_CURRENT,
              PCTPAGESSAVED_STATIC, AVGROWSIZE_STATIC,
              PCTPAGESSAVED_ADAPTIVE, AVGROWSIZE_ADAPTIVE
         FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('PAGECOMP', 'TAB_EXAMPLE'))
         WHERE OBJECT_TYPE = 'DATA';
```

- Result:

```
TABSCHEMA      TABNAME         ROWCOMPMODE ...
-------------- --------------- -----------
PAGECOMP       TAB_EXAMPLE                 ...

... PCTPAGESSAVED_CURRENT AVGROWSIZE_CURRENT ...
    --------------------- -----------------
...                     0                 805 ...

... PCTPAGESSAVED_STATIC AVGROWSIZE_STATIC PCTPAGESSAVED_ADAPTIVE AVGROWSIZE_ADAPTIVE
    -------------------- ----------------- ----------------------- ------------------
...                    78               171                      92                 47
```



Catalog Changes – SYSCAT.TABLES

- Compression-related columns in SYSCAT.TABLES:

```
db2 => SELECT TABSCHEMA, TABNAME,
               COMPRESSION, ROWCOMPMODE,
               AVGCOMPRESSEDROWSIZE, AVGROWCOMPRESSIONRATIO,
               PCTROWSCOMPRESSED, PCTPAGESSAVED
         FROM SYSCAT.TABLES
         WHERE TABSCHEMA = 'PAGECOMP' AND TABNAME = 'TAB_EXAMPLE';
```

- Result:

```
TABSCHEMA      TABNAME         COMPRESSION ROWCOMPMODE ...
-------------- --------------- ----------- -----------
PAGECOMP       TAB_EXAMPLE               R           A ...

... AVGCOMPRESSEDROWSIZE AVGROWCOMPRESSIONRATIO ...
    -------------------- ----------------------
...                   48           +1.67708E+001 ...

... PCTROWSCOMPRESSED PCTPAGESSAVED
    ----------------- -------------
...      +1.00000E+002            92
```

New column, indicates row compression mode:
S – Static
A – Adaptive
Blank – No Compression

Columns maintained by Runstats, reflect overall compression ratio

92% pages saved, corresponds to overall compression ratio of 12.5x

New in DB2 10: Log Archive Compression

Log Archive Compression

```
UPDATE DB CFG … USING LOGARCHCOMPR1 ON
UPDATE DB CFG … USING LOGARCHCOMPR2 ON
```

- Automatically compresses log extents on-the-fly when they are moved to the archive
  - Requires log archiving to be enabled (archiving methods: DISK, TSM, or VENDOR)
  - Automatic, on-the-fly expansion when log extents are retrieved by DB2 during ROLLBACK processing and ROLLFORWARD recovery
- Can be enabled independently for primary and secondary archiving methods
  - Controlled via dynamic database configuration parameters
  - Current settings shown via **`GET DB CFG`** command
- Compression algorithm is the same as for (default) backup compression

Best Practices

10 Best Practices for DB2 10

1. Use standard row format
   - Row format for compressed tables has little impact on the storage consumption
2. Use estimation capabilities to gauge compression savings
   - Focus on largest tables first
   - Use new ADMIN_GET_TAB_COMPRESS_INFO
3. During first adoption, choose REORG or ADMIN_MOVE_TABLE to compress
4. When you REORG:
   - Start with the smallest tables, work your way up
   - Use a temporary table space for the REORG, and specify the RESETDICTIONARY option
5. When you use ADMIN_MOVE_TABLE:
   - Put your tables into new-style DMS table spaces to facilitate file system space reclamation
   - Use ADMIN_MOVE_TABLE to compress and move your tables
6. Reclaim disk space via ALTER TABLESPACE REDUCE MAX

10 Best Practices for DB2 10

7. Use large table spaces over regular ones

8. Use separate table spaces for tables with different compression settings
   - Perform table space backups
   - Apply backup compression for table spaces that contain uncompressed tables

9. Group correlated columns
   - Adaptive compression can very effectively compress long sequences of similar column values
   - Compression ratio improves

10. Cluster your data
    - Sort before load, so that row order respects similarity
    - Compression ratio improves when similar rows are stored in the same page

Verifying Compression

- Perform RUNSTATS, and check compression-related columns in SYSCAT.TABLES
  - PCTPAGESSAVED
    - Should be at or near the projected savings
  - PCTROWSCOMPRESSED
    - Should be at or near 100%
  - AVGCOMPRESSEDROWSIZE
    - Should be longer than minimum row size for table space

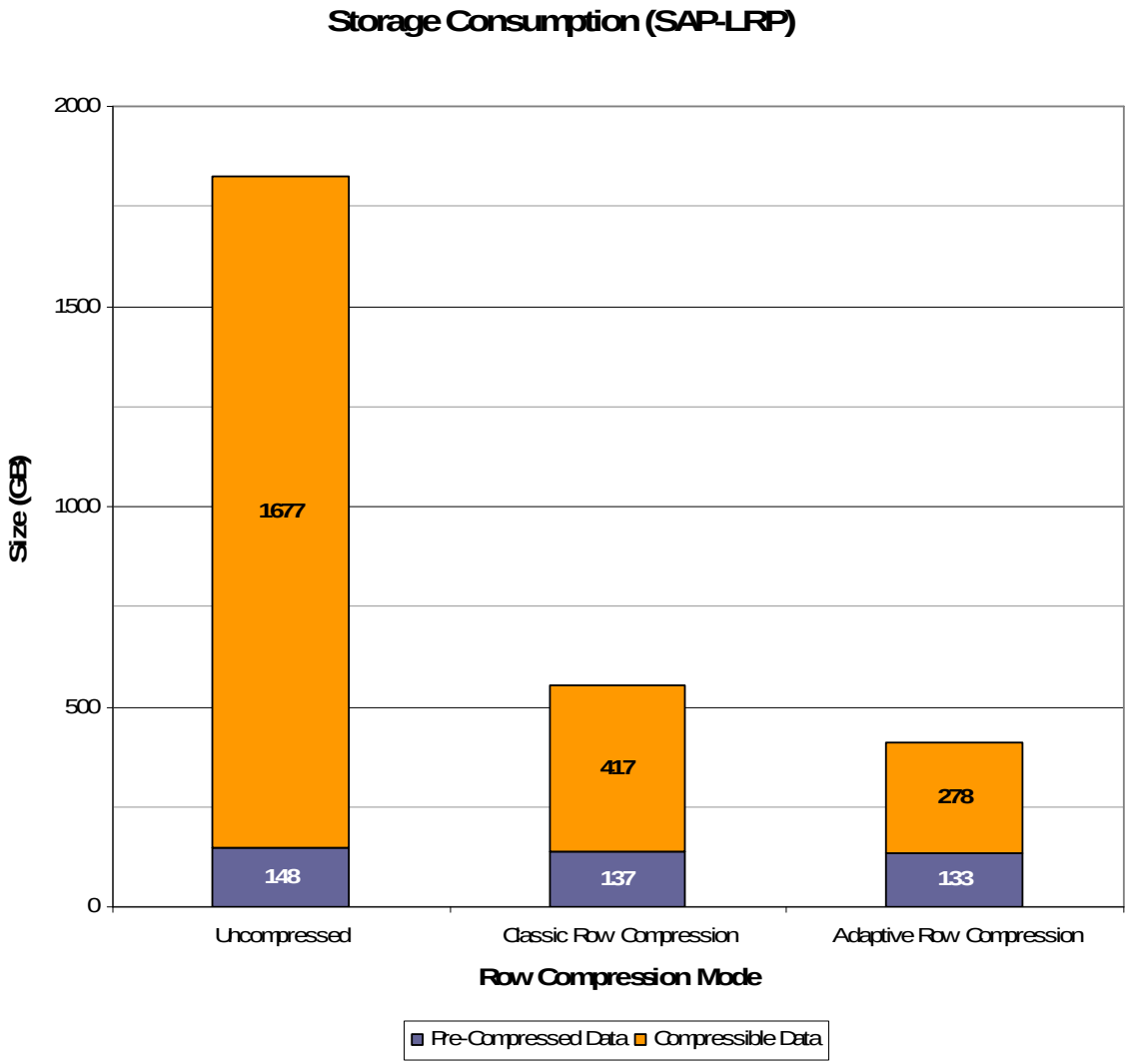
Common Pitfalls

- Compressed rows are shorter than minimum row length
  - Happens in REGULAR table spaces
  - Minimum row length is (roughly) pagesize divided by 255
  - Solution: Convert your table space to LARGE, or move table
- Automatic dictionary creation leaves uncompressed rows in table
  - When dictionary is created, existing rows are not compressed
  - Solution: Perform a table reorganization
- Table contains pre-compressed data
  - Most prominently, check for CHAR or VARCHAR FOR BIT DATA columns
- Compression may impact cell size calculations for MDC tables
  - Compressed rows are shorter, cells size will be reduced accordingly
  - This may result in a larger number of partial blocks
  - Solution: Reduce extent size, or modify dimensions (e.g. reduce or coarsify)

Performance

Storage Space Savings

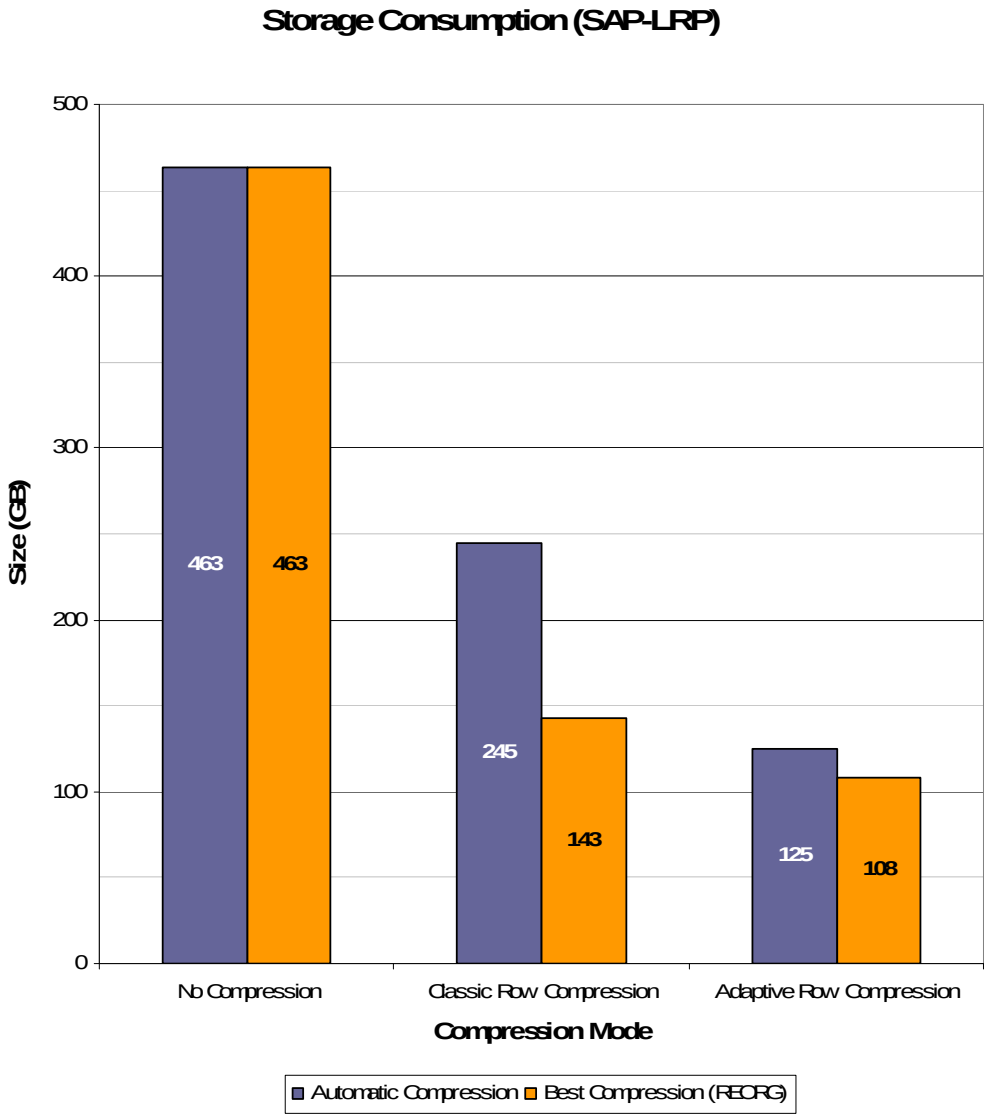
- SAP-LRP workload
- Size ~1.8TB uncompressed
- Considered 1425 largest tables (~99.5% total data)
- Best compression with classic row compression: **4x (75% savings)**
- Best compression with adaptive row compression: **6x (83% savings)**
- Improvement: **33%**
- Note: Size of pre-compressed data does not change



Reorg Avoidance – ADC vs. Classic Table Reorg

- SAP-LRP system with ~450GB of data in uncompressed form
- ~24,000 tables, 99.7% of the data in 1,500 tables
- Comparison between database build (ADC) and full Classic Table Reorg

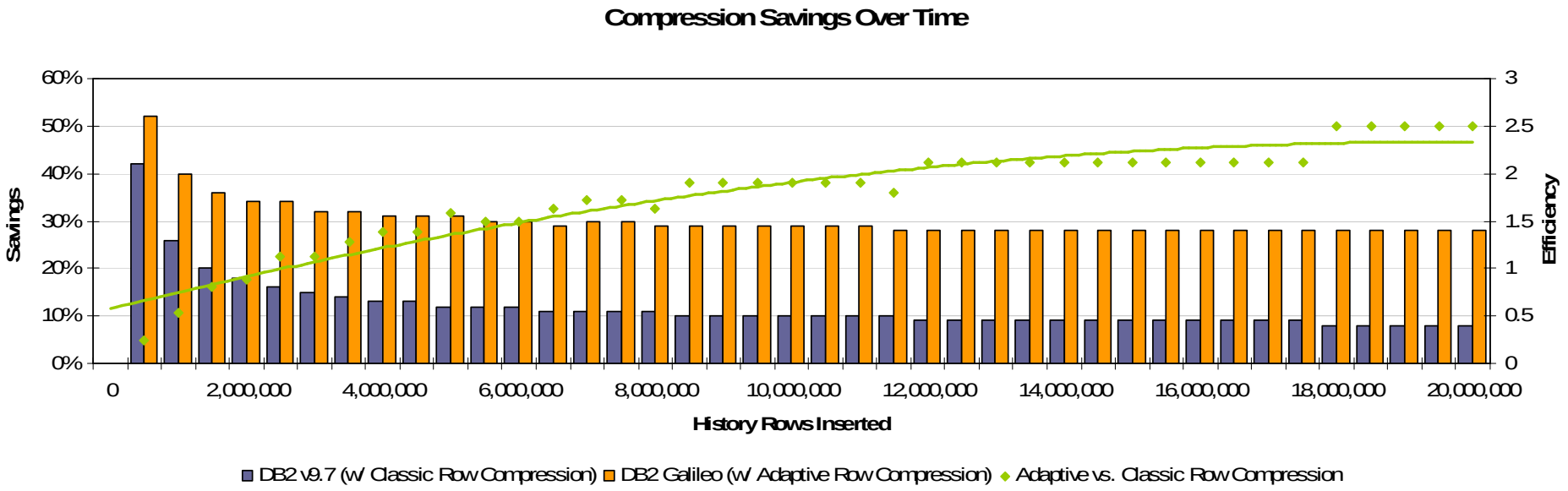
- Classic Row Compression with ADC yields **71% larger** data size than after Reorg
- Adaptive Compression with ADC is only **16% larger** than after Reorg
- Adaptive Compression with ADC is **14% smaller** than Classic Row Compression after Reorg (i.e. best possible compression)



Reorg Avoidance – More Stable Compression Ratio Over Time

- History table for ORDERS table (TPC-H)
- 20 million update operations
  - Partial row updates, random content
- History table has ADC table-level dictionary

- Classic Row Compression
  - Initially 42% storage space savings
  - Drops to <10% savings over time
- Adaptive Row Compression
  - Initially 52% storage space savings
  - Levels out at around 30% savings
- Adaptive Row Compression gets more efficient as Classic Row Compression degrades



Query and DML Performance

- Compression means additional CPU consumption for compression & expansion operations
- Row compression design paradigm: Compression under logical I/O
- Rule of thumb: Compression is beneficial in I/O-bound situations

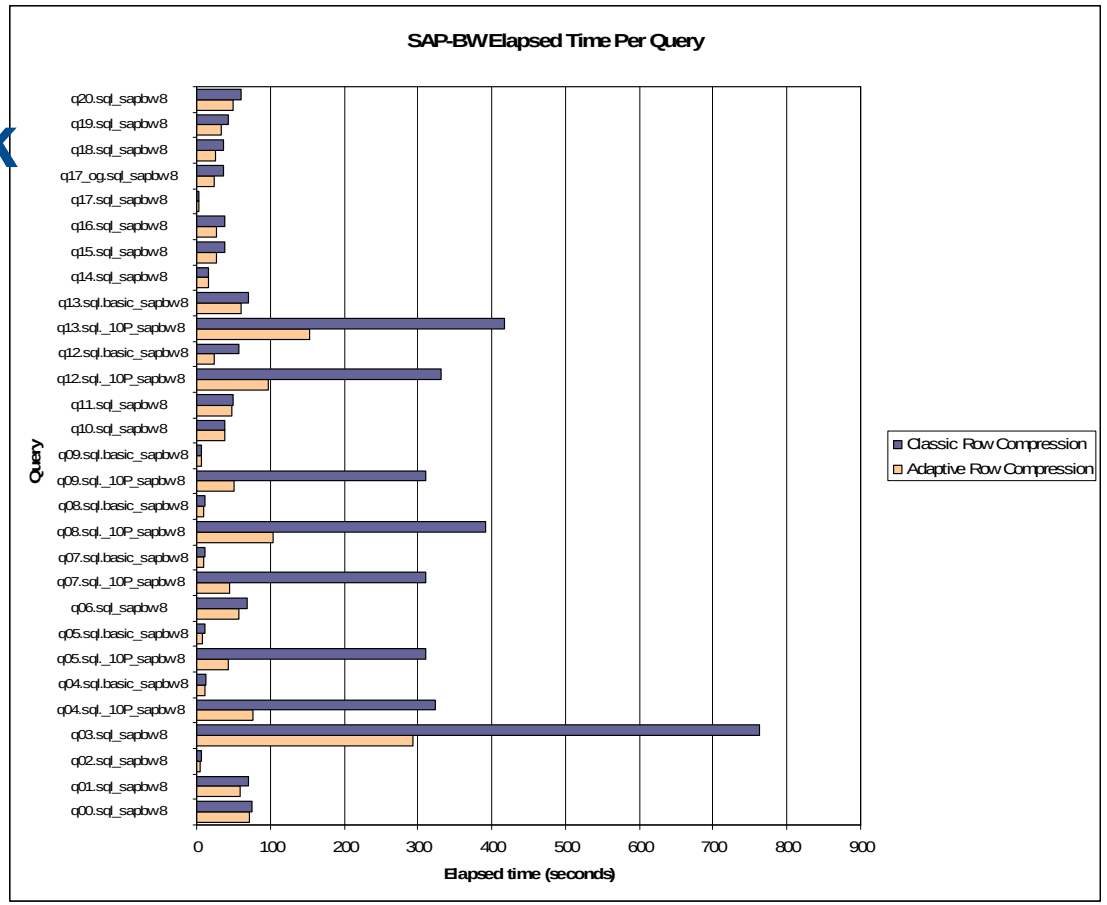
- Performance measurements done from two angles:
  1. Focus on additional CPU cost (to minimize overhead)
  2. Elapsed time studies in benchmark scenarios

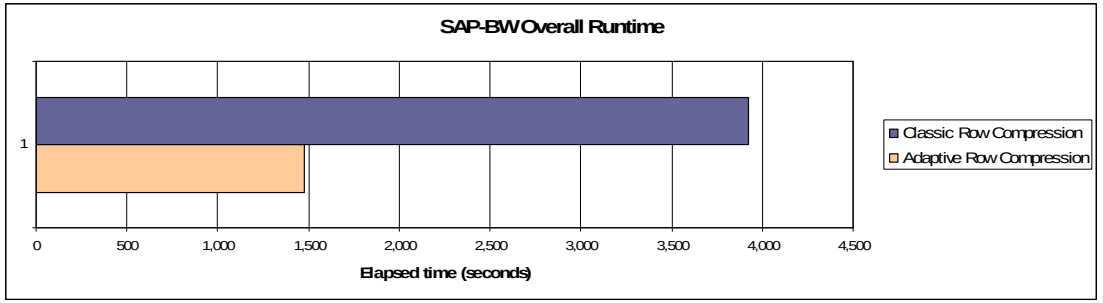
- CPU consumption almost always increases:
  - Fully buffered table scan: 20% to 25% overhead
  - Fully buffered single-row index lookup: 5% overhead
  - Complex queries: Between 60% improvement and 12% overhead
  - Insert: 5% to 15% overhead
  - Deletes: Up to 20% improvement
  - Updates: 15% overhead for few-row updates, higher for bulk operations

- **In many practical scenarios this translates into performance improvements**

Query Performance SAP-BW Benchmark

- SAP Business Warehouse benchmark (1TB database)
- Most long-running queries experience dramatic speed-up
- Overall runtime reduced by 62%
- Median query speedup of 43%





Thomas Fanghaenel
IBM
fanghaen@us.ibm.com

H06
Deep Dive Into Storage Optimization