White Paper

Huffman Data Compression in IBM Db2 for z/OS

Overview

With Db2 12 for z/OS function level 504 (FL504), the capability has been delivered to exploit the Huffman compression feature introduced in z14. This whitepaper will explain Huffman compression, its benefits, and how to enable it. This paper will also discuss the impact on the Broadcom® Database Management Solutions for Db2 for z/OS software.

Introduction

The target audience for this document are people who are involved in Db2 z/OS administration and data processing in general. Compression of data is an important technique to increase the capacity of your direct access storage device (DASD). When you compress data, you need less DASD space for your Db2 tables.

Data compression will improve I/O performance through the reduction of DASD utilization, and can benefit Db2 buffer pool performance resulting in lower CPU utilization. Over time, IBM has delivered numerous enhancements in the compression space for Db2 z/OS that have reduced the compression overhead, making it more attractive for customers to utilize.

With FL504, Huffman compression uses IBM z14 hardware to compress data before it is stored and to decompress the data that is retrieved from a page in the buffer pool.

Huffman compression, in general, will provide better compression ratios, but may have a higher CPU cost than fixed-length for decompression. We will explain more about that later, but first let us start with an overview of compression capabilities on z/OS and its exploitation by Db2.

Compression Support Options on z/OS

The development of Db2 data compression support has progressed over the course of the product lifecycle. Initially, the only way to compress Db2 table data was to use an EDITPROC, a software algorithm high in the computational stack. The cost of compression and decompression was often prohibitive.

In the Db2 V3 time-frame, IBM introduced the CMPSC instruction. Initially, the instruction was implemented in millicode. So, the operation was brought down significantly in the stack of computational layers. This capability reduced the cost of compression and decompression and made the larger scale compression of Db2 tablespaces possible.

The z800 and z900 machine models were the first to implement the CMPSC instruction in a special hardware co-processor. The hardware implementation offers considerably better performance than what was available on earlier systems.

Processors prior to the z14 had hardware support for only a Lempel/Ziv type of compression. The z14 introduced Huffman compression. The z15 introduced on-chip Gzip compression. Before the z15, Gzip compression was a separate feature on the zEDC express (PCIe) card. With the z15, it is now an on-chip accelerator, avoiding the overhead of data movement to and from the PCIe card. The z15 on-chip accelerator reportedly boosts compression throughput from 1 to 16 GB per second.

As these hardware improvements became available, the Db2 engine exploited z-synergy strategies and made internal improvements, as well. All of these combined compression support options resulted in improved compression performance over the course of the various Db2 releases.

Db2 for z/OS Compression

Various elements of a Db2 subsystem can be compressed. However, the only compression type that is affected by the new Huffman compression support is the dictionary-based compression as executed by the CMPSC instruction.

Tablespaces

Data compression for Db2 started with tablespace compression. Initially this was through software, but IBM quickly brought hardware support in the form of the CMPSC instruction. The CPMSC instruction takes a piece of data (a row) and compresses or decompresses it using a dictionary. While compression is enabled at the tablespace partition level, a partition may contain a combination of compressed and uncompressed rows. Before rows can be compressed, the compression dictionary must first be constructed. An important characteristic of tablespace compression is that the data is compressed in the bufferpool.

This type of compression also applies to XML data.

The compression dictionary can be rebuilt by executing a REORG or LOAD REPLACE utility of a partition. The main difference between REORG and LOAD REPLACE is the number of rows used to build the dictionary.

Lob Data

With Db2 12, LOB data can be compressed, as well, using zEDC compression and not using a compression dictionary. Since this paper is on Huffman compression, LOB data compression is beyond its scope.

Indexspaces

Indexspaces can also be compressed in order to save DASD space. The compression is dictionary-less and performed by the Db2 I/O engines. The page in the bufferpool is, therefore, in an uncompressed format. As a result, there is no impact on either the bufferpool hit ratio or on virtual storage savings.

The Db2 deferred write engine asynchronously compresses a leaf page from a buffer into an I/O work area and then writes the page to disk. Conversely, pages are read from the disk into the I/O work area and then expanded into a buffer, in the buffer pool. The expansion is synchronous if the I/O itself is synchronous and the expansion is asynchronous if the I/O itself is asynchronous. All synchronous CPU time is counted as class 2 CPU time. All asynchronous CPU time is counted as DBM1 SRB time (which is zIIP eligible).

Log Data

Db2 archive log datasets can be compressed by a data facility storage management subsystem (DFSMS). With the zEDC feature (either as a separate card or by exploiting the compression core on Z15 machines), archiving log datasets has become an attractive option. Compressed archive datasets allow more log data to be available on expensive DASD, optimizing a log intensive recovery.

Compression Method for Db2 Tablespaces

There are many compression methods. For database systems such as Db2, the method must allow the original data to be perfectly reconstructed from the compressed data so that after compression and decompression, the data is identical to the original data. Lempel/Ziv or Huffman compression methods are the most commonly used of the general purpose compression methods. In their simplified essence, both methods replace a string of data with a shorter representation.

Db2 builds or rebuilds the dictionary during execution of the LOAD utility or REORG utility. Since Db2 11, a compression dictionary can also be built dynamically when adding a sufficient amount of data to a tablespace, defined with COMPRESS YES. A compression dictionary can be built during SQL INSERT or MERGE, or execution of a LOAD RESUME utility.

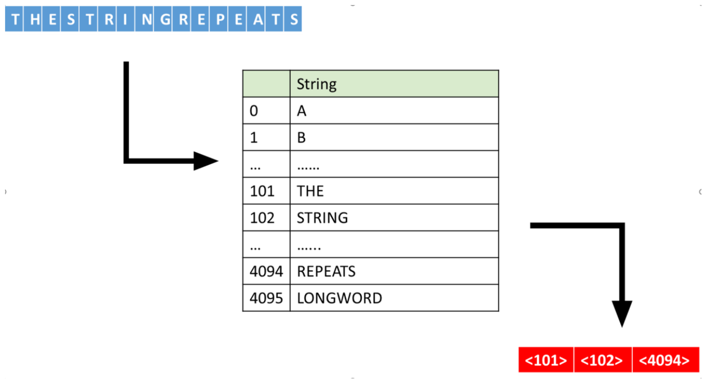
What is Lempel/Ziv Compression?

Both Lempel/Ziv and Huffman use a dictionary to support compression and decompression of the data. The variation of Lempel/Ziv used by Db2 (and supported by CMPSC) is LZ78. In LZ78, the shorter representation is actually a pointer to the entry in the decompression dictionary.

The dictionary can be seen as a table with the entries representing the data that has been replaced by a shorter code. The shorter code is stored in the compressed image of the row. Typically, Db2 has 4,096 entries in the dictionary. When building this dictionary, Db2 attempts to optimize the compression ratio by keeping the most frequently used byte strings as entries in the dictionary. As an example, the effect of replacing a 12 byte string by a 12 bit pointer only once in a 25 MB dataset is hardly measurable. However, replacing a 6 byte string a million times by a 12 bit pointer in the same 25 MB dataset will shrink it significantly.

In the Figure 1 example, a 16 byte (128 bits) string gets reduced to three index entries in the table containing 4,096 entries. The index entry could fit in 12 bits. This results in 128 bits being reduced to 36.

Figure 1: Lempel/Ziv Compression Example



	String
0	A
1	B
...	......
101	THE
102	STRING
...	......
4094	REPEATS
4095	LONGWORD

While the example in Figure 1 is severely oversimplified, it demonstrates the principle of Lempel/Ziv compression.

What is Huffman Compression?

Just like Lempel/Ziv, Huffman is a dictionary-driven compression and decompression algorithm. However, the entries in the dictionary are sorted by frequency of occurrence. The entries with the highest frequency get a shorter bit pattern to identify the entry in the dictionary. This typically results in a much improved compression ratio.

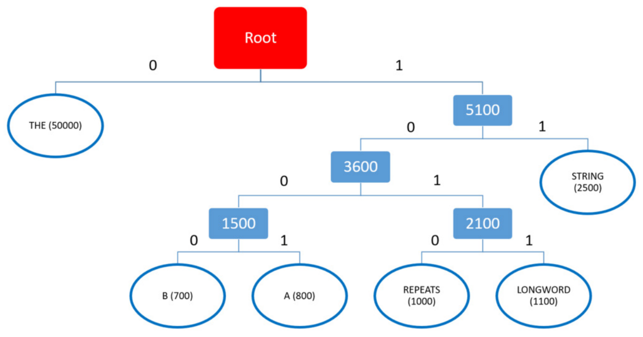
The best way to visualize Huffman compression is to imagine the entries being organized in a tree-like structure. This is not imaginary since this is actually the way Huffman compression algorithms build the dictionary. In order to achieve this, in addition to gathering the entries as for a Lempel/Ziv type dictionary, the selected entries are sorted in ascending frequency order.

Figure 2: Entries from the Lempel/Ziv Compression Example with Added Imaginary Frequencies

String	Frequency
A	800
B	700
......	....
THE	50000
STRING	2500
......	....
REPEATS	1000
LONGWORD	1100

An algorithm uses the frequencies of the entries and gives the shortest bit pattern to the entry with the highest frequency. To support this, a tree-like structure is built.

Figure 3: Tree-Like Structure Based on the Frequencies of the Entries



In the previous figure of the tree-like structure, the oval shapes represent the entries. Each entry carries the value it represents as well as the frequency of the entry. The rectangular shapes represent intermediate nodes that represent the frequency of the subtree structures (or nodes) below them.

The bit pattern that is associated with each entry can be determined by following the path from the root towards the entry and concatenating the bits of the branches taken.

Figure 4: Resulting Bit Patterns from the Sample in Figure 3

String	Frequency	Bit pattern
A	800	1001
B	700	1000
……	….	
THE	50000	0
STRING	2500	11
……	….	
REPEATS	1000	1010
LONGWORD	1100	1011

Using these bit patterns, the entry with the highest frequency ('THE') will be represented with a single bit instead of the 12 bit code as in the Lempel/Ziv example. This demonstrates the important distinction between Huffman and Lempel/Ziv and explains the values of the ZPARM TS_COMPRESSION_TYPE 'HUFFMAN' compared to ZPARM TS_COMPRESSION_TYPE 'FIXED_LENGTH'.

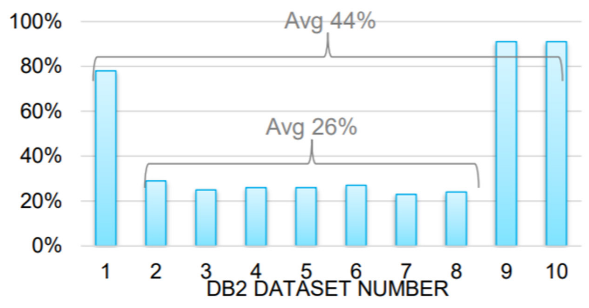
Why Huffman Compression?

Huffman compression may achieve better compression ratios in general since it assigns the shortest bit patterns to the symbols with the highest frequency. For more information, refer to the IBM Db2 z/OS documentation. As is usual with Db2 dictionaries, the method through which they are built may have an impact on the compression characteristics. Only dictionaries built by the REORG utility have an opportunity to sample the data of the entire partition. Using the REORG utility typically gives the best compression ratio.

Early IBM lab tests have shown that Huffman typically compresses between 25% and 45% better than Lempel/Ziv. However, of course, your results may vary.

The following figure, comes from a presentation by Dr. Christian Jacobi (Chief Architect, IBM Z processor development). The diagram depicts the achieved improvements in compression ratio. The range in the tests shown here range from 20% to 80% better compression.

Figure 5: Disk Reduction from Huffman Coding Compared to Traditional Compression Methods



Deciding Whether to Compress a Tablespace Partition

The primary question to answer is, "Do I compress or not?" In preparing this paper, we asked our Db2 customers about their decision criteria for compression. Some customers answered that they compress by default. Others have some size limits that need to be met before they decide to compress a tablespace; the decision is made either by developers or is reviewed on a regular basis. No respondent mentioned actively monitoring the cost of compression and decompression.

When Does It Make Sense to Compress?

The process of compression and decompression results in processing overhead. Db2 z/OS has come a long way with enhancements involving both hardware and software synergies to reduce this overhead. Even with these enhancements there may still be some measurable compression overhead.

The purpose of compression is to achieve savings by reducing the number of data pages needed for the tablespace. Since page size on DASD is not affected by tablespace compression, the reduction is achieved only by packing more rows on a single page. It is clear that reducing the number of pages will have a positive effect on the amount of I/O needed to process the tablespace and save the CPU to drive the I/O. The process of taking an image copy is a good example of this. The reduction of pages results in direct savings because taking an image copy does not now require decompression.

However, there are examples where the effect of compression is negated. The fundamental principle of reducing the number of pages that are required to house the rows by fitting more rows onto a single page may not be achieved in some contexts, such as the following instances:

- MAXROWS is used to limit the number of rows per page.
- The hard 255 row per page limit is in effect.
- Rows do not compress enough to allow significantly more rows per page.

In these cases, the space savings are not achieved and compression equals pure overhead costs.

Then there will be data access to consider. SQL UPDATE will have to decompress and compress the rows that are updated; therefore, tablespaces that will have a heavy update workload will be less attractive to compress than tablepaces that are predominantly INSERT/SELECT.

Going a step deeper, having more rows on each page, in principle, means less getpages (direct CPU savings). When the clustering of the tablespace supports the processing pattern, the bufferpool hit ratio will also improve. An improved bufferpool hit ration will provide even more CPU savings. However, there are exceptions to this rule:

- The SQL statements have a high rows scanned to rows qualified ratio.
  - The more the rows are spread across pages, the more important this gets.
- The SQL statements do not have to scan many rows (perhaps even direct access) but the bufferpool hit ratio does not change significantly.
  - Hence, there is no real savings in I/O.

Such data access criteria are difficult to use as a general rule. Especially in installations with many tablespaces. The mix is just too big. However, for exceptional cases, an administrator might conclude that the cost of a highly used and important SQL statement is affected negatively by compression.

Restricted Number of Rows for Each Page

The hard limit of the number of rows per page comes from the RID. Even though Db2 12 now supports up to seven byte RIDs (for RPN tablespaces), the increase in RID length was meant to accommodate more data pages. The part in the RID that identifies the row within the page is still 1 byte. The net effect is that there can be no more than 255 rows in a single page, regardless of the page size. So, if the rows are so short that even if they remained uncompressed, 255 rows would still fit in your chosen page size; and there is absolutely no savings. Thus, in this case, the overhead that it takes to compress the tablespace does not pay off.

There are also cases where Db2 customers have chosen to limit themselves by using the MAXROWS clause of the CREATE TABLESPACE. This is typically done to reduce lock contention while avoiding the page p-lock overhead that is associated with Row Level Locking in data sharing. The extreme case is tablespaces with MAXROWS=1, effectively mimicking Row Level Locking. Of course, for tablespaces such as these, data compression will only have negative effects.If you use page level locking, then the span of a lock is all the data on the page. Compression increases the span of a lock, and Huffman compression can increase that further. At the very least, customers should be aware of and monitor locking as they move to Huffman. The industry best practice is to not compress tablespaces defined with MAXROWS=1.

Huffman Compression Prerequisites

A few requirements must be fulfilled in order to reap the benefits of Huffman compression. First and foremost, your Db2 must be at V12, functional level 504 or higher (APAR PH04424, PTF UI60685). This functional level delivers the new code in Db2 to build and use Huffman dictionaries.

Further, Huffman compression is ONLY available for UTS tablespaces.

The Huffman dictionaries are only understood by the CMPSC instruction (compression co-processor) of z14 and higher hardware. Db2 will only build Huffman dictionaries when it is executing on hardware with the required support. If a Huffman compressed tablespace is accessed from a z13 machine then de-compression will be performed by software emulation. Of course, this will cause significant CPU overhead.

There are cases where compressing (or using Huffman to compress more) can negatively affect an application that is using page level locking. Do not compress tablespaces defined with MAXROWS=1.

Implementing Huffman Compression

The value of ZPARM TS_COMPRESSION_TYPE that is in control when a tablespace partition is compressed will determine the resulting dictionary type. This makes it very hard to manage at an object level in a true productive environment. Currently, Db2 does not allow specification of the dictionary type at the object level. The easiest way forward would be to test Huffman in a test environment, validating both the compression improvements and the CPU differences for typical workloads. Based on this analysis, you can decide on a zParm setting for the production environments.

In a data sharing environment, you could theoretically have different members with different values for TS_COMPRESSION_TYPE. The member where the dictionary is created (typically the LOAD/REORG without keep dictionary) would then determine the compression algorithm used. In a data sharing implementation, a best practice is to give all members the same zParm setting for compression.

As previously stated, the requirements to support Huffman compression for each member are the same.

A Huffman dictionary is only generated for LOAD, REORG, and compress on INSERT under the following conditions:

- Hardware support is present (z14 or later) on the member on which the dictionary is created.
- Function level 504 or higher is activated.
- zParm TS_COMPRESSION_TYPE is set to HUFFMAN.
- The tablespace is a universal tablespace (UTS).

If any of the above conditions are not met, Db2 will generate the classic fixed-length dictionary to be used by Lempel/Ziv. There is no error message indicating that there is a conflict between members for the zParm setting. And there is no hard failure. The process simply continues on as before with Lempel/Ziv compression. This process could produce undesired results in a data sharing infrastructure. Some of these undesired results might include:

- CPU cost incurred while building a new dictionary
- Differences in runtime for LOAD or REORG
- The possibility of reduced compression percentage

If data is compressed using a Huffman encoded dictionary and then moved to a z13 or older system, then data can still be expanded, but it is done with software, thus resulting in increased resource utilization.

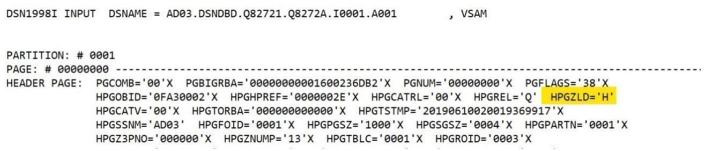
Certainly, there will be some customers who might disregard best practices and utilize system affinity in some form. This is problematic for a variety of reasons, all of which are well known, including the simplest and most detrimental reason, which is human error. This type of implementation would increase the administrative burden for job execution, and disable some of the advantages of utilizing data sharing. Therefore, this type of implementation would be a problematic endeavor.

How Do I Know Which Algorithm Was Used to Compress the Data?

The only way to verify the dictionary type is to execute a DSN1PRNT and look at the value for HPGZLD. There is no indication in the Db2 catalog table.

A value of 'H' (such as in the following figure) indicates a Huffman dictionary. A value of 'L' is used for Lempel/Ziv (fixed length) dictionaries. This value designation is also how Db2 knows how to set up for the CMPSC instruction.

Figure 6: A Value of 'H' Indicates a Huffman Dictionary



```
DSN1998I INPUT  DSNAME = AD03.DSNDBD.Q82721.Q8272A.I0001.A001       , VSAM

PARTITION: # 0001
PAGE: # 00000000 -------------------------------------------------------------------------
HEADER PAGE:  PGCOMB='00'X  PGBIGRBA='00000000001600236DB2'X  PGNUM='00000000'X  PGFLAGS='38'X
              HPGOBID='0FA30002'X  HPGHPREF='0000002E'X  HPGCATRL='00'X  HPGREL='Q'  HPGZLD='H'
              HPGCATV='00'X  HPGTORBA='000000000000'X  HPGTSTMP='20190610020019369917'X
              HPGSSNM='AD03'  HPGFOID='0001'X  HPGPGSZ='1000'X  HPGSGSZ='0004'X  HPGPARTN='0001'X
              HPGZ3PNO='000000'X  HPGZNUMP='13'X  HPGTBLC='0001'X  HPGROID='0003'X
```

Using DSN1COMP to Assess Savings

While IBM tests have indicated that Huffman compresses 25% to 45% better than Lempel/Ziv, your results may vary. In order to provide a better understanding of the effectiveness of Huffman compression for an object, the DSN1COMP standalone utility has been enhanced to support this assessment.

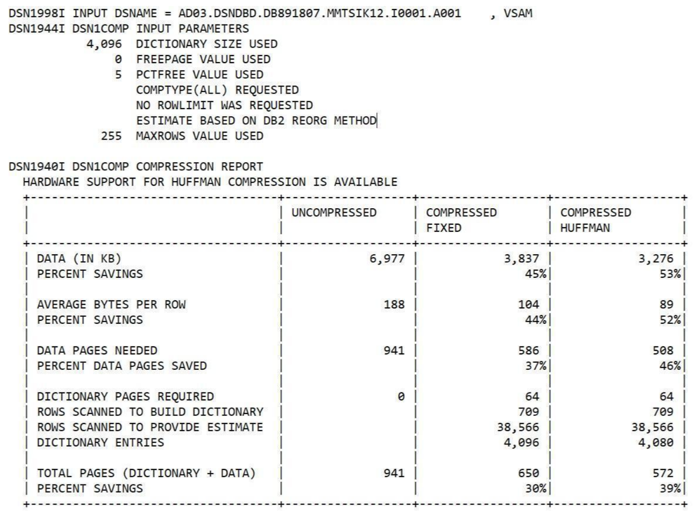
PH19242 (UI6938) added support for Huffman compression to the DSN1COMP standalone utility. The APAR adds the COMPTYPE keyword to DSN1COMP. The keyword takes any of the values shown in the following table.

Table 1: Keyword Values

Key Word	Value
FIXED	DSN1COMP gives an estimate using a fixed length (traditional) compression dictionary.
HUFFMAN	DSN1COMP gives an estimate using a fixed length (traditional) compression dictionary.
ALL	DSN1COMP gives an estimate of both fixed length and Huffman compression dictionary. This allows a comparison of both methods.

The following report was generated with 'COMPTYPE(ALL)' to show there can be a significant improvement in the compression ratio.

Figure 7: COMPTYPE(ALL) Report



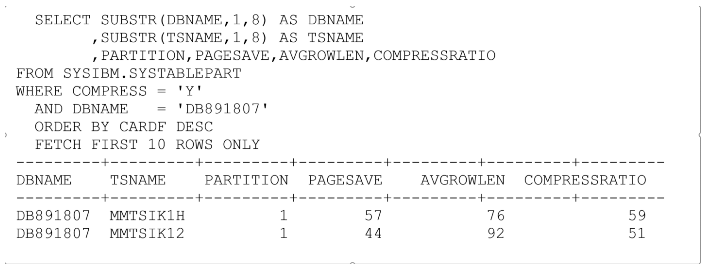
```
DSN1998I INPUT DSNAME = AD03.DSNDBD.DB891807.MMTSIK12.I0001.A001    , VSAM
DSN1944I DSN1COMP INPUT PARAMETERS
          4,096  DICTIONARY SIZE USED
              0  FREEPAGE VALUE USED
              5  PCTFREE VALUE USED
                 COMPTYPE(ALL) REQUESTED
                 NO ROWLIMIT WAS REQUESTED
                 ESTIMATE BASED ON DB2 REORG METHOD
            255  MAXROWS VALUE USED

DSN1940I DSN1COMP COMPRESSION REPORT
  HARDWARE SUPPORT FOR HUFFMAN COMPRESSION IS AVAILABLE
  +------------------------------------+------------------+------------------+------------------+
  |                                    | UNCOMPRESSED     | COMPRESSED       | COMPRESSED       |
  |                                    |                  | FIXED            | HUFFMAN          |
  +------------------------------------+------------------+------------------+------------------+
  | DATA (IN KB)                       |            6,977 |            3,837 |            3,276 |
  | PERCENT SAVINGS                    |                  |              45% |              53% |
  |                                    |                  |                  |                  |
  | AVERAGE BYTES PER ROW              |              188 |              104 |               89 |
  | PERCENT SAVINGS                    |                  |              44% |              52% |
  |                                    |                  |                  |                  |
  | DATA PAGES NEEDED                  |              941 |              586 |              508 |
  | PERCENT DATA PAGES SAVED           |                  |              37% |              46% |
  |                                    |                  |                  |                  |
  | DICTIONARY PAGES REQUIRED          |                0 |               64 |               64 |
  | ROWS SCANNED TO BUILD DICTIONARY   |                  |              709 |              709 |
  | ROWS SCANNED TO PROVIDE ESTIMATE   |                  |           38,566 |           38,566 |
  | DICTIONARY ENTRIES                 |                  |            4,096 |            4,080 |
  |                                    |                  |                  |                  |
  | TOTAL PAGES (DICTIONARY + DATA)    |              941 |              650 |              572 |
  | PERCENT SAVINGS                    |                  |              30% |              39% |
  +------------------------------------+------------------+------------------+------------------+
```

DSN1COMP only works with objects that are not currently compressed. Therefore, DSN1COMP cannot be used to assess the potential impact of switching from traditional compression to Huffman compression.

Compressing the Tablespace with REORG

The tablespace used in the following example of DSN1COMP is for a duplicate tablespace. (An exact copy was made of existing tablespace.) The tablespace used in the following example of DSN1COMP is for a duplicate tablespace.

Figure 8: Example of DSN1COMP



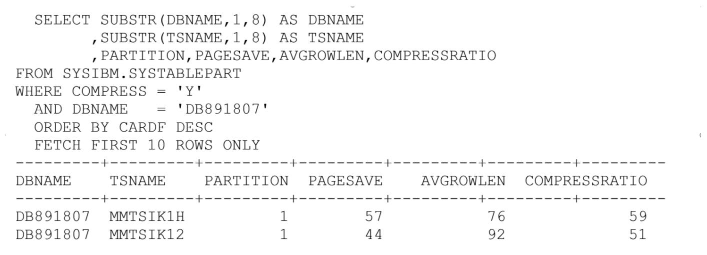
```
  SELECT SUBSTR(DBNAME,1,8) AS DBNAME
        ,SUBSTR(TSNAME,1,8) AS TSNAME
        ,PARTITION,PAGESAVE,AVGROWLEN,COMPRESSRATIO
FROM SYSIBM.SYSTABLEPART
WHERE COMPRESS = 'Y'
  AND DBNAME    = 'DB891807'
  ORDER BY CARDF DESC
  FETCH FIRST 10 ROWS ONLY
---------+---------+---------+---------+---------+---------+---------
DBNAME    TSNAME     PARTITION  PAGESAVE    AVGROWLEN  COMPRESSRATIO
---------+---------+---------+---------+---------+---------+---------
DB891807  MMTSIK1H           1        57           76             59
DB891807  MMTSIK12           1        44           92             51
```

The following procedure was executed using REORG to compress both tablespaces:

1. Verify current setting for TS_COMPRESSION (it was HUFFMAN).
2. REORG tablespace MMTSIK1H (Huffman).
3. Set TS_COMPRESSION to FIXED.
4. REORG tablespace MMTSIK12.
5. Set TS_COMPRESSION back to what it was previously.

After REORG was used to compress both tablespaces, the catalog now shows the results shown in the following figure:

Figure 9: Catalog Results



```
  SELECT SUBSTR(DBNAME,1,8) AS DBNAME
        ,SUBSTR(TSNAME,1,8) AS TSNAME
        ,PARTITION,PAGESAVE,AVGROWLEN,COMPRESSRATIO
FROM SYSIBM.SYSTABLEPART
WHERE COMPRESS = 'Y'
  AND DBNAME    = 'DB891807'
  ORDER BY CARDF DESC
  FETCH FIRST 10 ROWS ONLY
---------+---------+---------+---------+---------+---------+---------
DBNAME    TSNAME     PARTITION  PAGESAVE    AVGROWLEN  COMPRESSRATIO
---------+---------+---------+---------+---------+---------+---------
DB891807  MMTSIK1H           1        57           76             59
DB891807  MMTSIK12           1        44           92             51
```

In this particular example, Huffman leads to better compression.

Broadcom Support

The Database Management Solutions for Db2 for z/OS provide toleration support of Huffman compression. The components of the solution have the ability to process Db2 objects already compressed using the Huffman compression algorithm. The support is delivered with PTF SO14029 (base install, with co-requisites for installed products).

To implement Huffman compression in your Db2 environments, to build compression dictionaries or to rebuild compression dictionaries for Db2 objects already compressed using the Huffman Compression algorithm, Broadcom Db2 Utilities (Rapid Reorg and Fast Load) calls the equivalent IBM/Rocket Utilities in the background.

Adoption of Huffman compression is still evolving as customers begin to utilize Db2 FL504. As you first begin to adopt Huffman compression, ensure that you are current on your maintenance for z/OS, Db2 for z/OS, and your Broadcom portfolio in respect to Huffman.

Sample JCL Delivered by Broadcom

For the Database Management Solutions for Db2 for z/OS, three sample JCLs are provided in customer_HLQ. CDBAJCL that you can use to overcome the fact that DSN1COMP cannot run against a tablespace that is already compressed.

Table 2: Provided Sample JCL

JCL Number	Function
INSHUFFC	Create a Db2 repository for DSN1COMP statistics
INSHUFFS	Run report, take data sample using SQL
INSHUFFU	Run report, take data sample using utilities

Both the INSHUFFS and INSHUFFU JCLs load sample data from the identified target table into an uncompressed tablespace and then run the IBM DSN1COMP standalone utility. INSHUFFS uses SQL INSERT INTO / SELECT FROM to get a sample of the data into a temporary object. INSHUFFU uses IBM utilities UNLOAD/LOAD to sample the data. If the sample size is large, it is less efficient to use the SQL version. Typically, you can achieve accurate results with small samples of 3% to 5%. In addition, the number of rows processed by DSN1COMP can be limited with this keyword: PARM='ROWLIMIT'.

INSHUFFS/INSHUFFU follows a logical procedure, such as this one:

1. Check for compression on a target tablespace; set RC=4 uncompressed RC=0 compressed.
2. If the target is not compressed, skip to a DSN1COMP step directly on the target tablespace.
3. If the target is compressed, follow these steps:
   a. Create a temporary tablespace (with user supplied variables) and a table using the create table command such as target table.

b. Unload a sample of data from the target table.

c. Load this sample data into the temporary table.

d. Run DSN1COMP against the temporary tablespace.

e. Drop the temporary tablespace.

f. Create a CSV file from the DSN1COMP output (REXX exec on hlq.CDBACLS0).

g. Load the CSV into the Db2 repository created by INSHUFFC.

Create the Repository with Statistics

All three examples in the following section require that you specify the following parameters.

Table 3: Parameters Common to All JCL

Parameter	Description
SSID	The Db2 sub-stem name or the group attach name of the data-sharing group.
DB2EXIT	The Db2 SDSNEXIT library.
DB2LOAD	The Db2 SDSNLOAD library.
TEP2PLAN	The plan name of the DSNTEP2 program in use.
TEP2LOAD	The name of the library containing the DSNTEP2 load module.

Before you use the INSHUFFS and INSHUFFU JCL, you must run the INSHUFFC JCL to create the repository table. The repository table contains the relevant statistics for all objects that were processed using the INSHUFFS or INSHUFFU JCL.

Table 4: Additional JCL Parameters for INSHUFFC

Parameter	Description
CREATE	Create repository objects (1 - Yes, 0 - No).
TRUNCATE	Truncate repository objects (1 - Yes, 0 - No).
DBNAME	The database in which the repository will be created. NOTE: the database will be dropped.
TSNAME	The tablespace to contain the repository table.
REPOSTB	The name of the repository table. Specify the fully qualified name.
BPOOL	The bufferpool for the repository tablespace.
STOGROUP	The storage group name for the repository tablespace.

Selecting Objects of Interest

The rule of thumb (as described by IBM) says that with compression rates below 10% or 20%, compression overhead could outweigh the savings achieved with compression. Huffman compression could push compression rates that are lower than 10% or 20% to above this ROI threshold.

The following example SQL statement gives a list of partitions that are currently uncompressed and might benefit from compression due to their size.

Figure 10: Example Selects Partitions Larger than 250 MB

```
select strip(tab.creator)
       concat '.' concat
       tab.name                  as tbname
      ,ts.maxrows
      ,ts.pctfree
      ,prt.partition
      ,max(prt.partition,1)  as part
      ,prt.vcatname
      ,prt.iprefix
      ,prt.dbname
      ,prt.tsname
      ,prt.bpool
      ,prt.spacef * 1024 as megabytes
      from sysibm.systablepart prt
inner join sysibm.systables    tab
   on  tab.dbname    = prt.dbname
   and tab.tsname    = prt.tsname
inner join sysibm.systablespace ts
   on  tab.dbname    = ts.dbname
   and tab.tsname    = ts.name
where prt.compress   = ' '
  and prt.spacef * 1024 > 250
 with ur;
```

The following example SQL statement gives a list of partitions that are already compressed but might still benefit from Huffman compression because their current compression ratio is low.

Figure 11: Example List of Partitions that are Compressed

```
select strip(tab.creator)
       concat '.' concat
       tab.name                  as tbname
      ,ts.maxrows
      ,ts.pctfree
      ,prt.partition
      ,prt.vcatname
      ,prt.iprefix
      ,prt.dbname
      ,prt.tsname
      ,prt.bpool
      ,prt.compressratio
      from sysibm.systablepart prt
inner join sysibm.systables    tab
   on  tab.dbname    = prt.dbname
   and tab.tsname    = prt.tsname
inner join sysibm.systablespace ts
   on  tab.dbname    = ts.dbname
   and tab.tsname    = ts.name
where prt.compress   = 'Y'
  and prt.compressratio < 20
 with ur;
```

Gathering the DSN1COMP Statistics for the Objects

The SQL statements in the previous example provide the data that INSHUFFS and INSHUFFU require.

Both INSHUFFS and INSHUFFU require the following parameters in addition to the parameters common to all.

Table 5: INSHUFFS and INSHUFFU Parameters

Parameter	Description
HLQ	High level qualifier of datasets internal to the job.
UNIT	Unit name for datasets internal to the job.
VCAT	High level qualifier of the VSAM cluster for DSN1COMP.
TARGDB	The database that contains the partition to be analysed.
TARGTS	The tablespace to be analysed.
TARGTB	The tablename contained in the tablespace to be analysed.
PART	The partition identifier in case the tablespace is uncompressed (part column in sample SQL).
TEMPTS	The temporary tablespace where the data from the target tablespace will be stored in case the tablespace (partition) is currently compressed. A table will be created. The tablespace will be created in TARGDB.
TEMPTB	The name of the table that is created in TEMPTS to temporarily hold the data.
PCTFREE	The percent free parameter for the temporary tablespace.
MAXROWS	The maximum row parameter for the temporary tablespace.
BPOOL	The bufferpool for the temporary tablespace.
STOGROUP	The storage group name for the temporary tablespace.
REPOSTB	The name of the repository table created in INSHUFFC.
REXXLIB	The name of the library containing the distributed REXX exec. *<customer HLQ>*. CDBACLS0.
ROWS	The number of rows to copy to the temporary object in case the target tablespace is compressed.
IJ	This parameter is taken from the iprefix column in the sample SQL.

About the Mainframe Division at Broadcom

The Mainframe Division at Broadcom continues to drive the next horizon of open, cross-platform, enterprise solutions. We specialize in DevOps, security, AIOps, and infrastructure software solutions that allow customers to embrace open tools and technologies, make mainframe an integral part of their cloud, and enable innovation that drives business forward. We are committed to forging deep relationships with our clients at all levels. We go beyond products and technology to partner with you in creative ways that support your success.

Authors

Toine Michielse, Client Services Consultant
Toine Michielse has been working with Db2 for z/OS ever since the ESP of V1. In his career, he has worked as a COBOL/IMS/Db2 programmer, and as an IMS/Db2 DBA and Db2 system engineer. He has worked for a number of years as a Db2 Lab advocate, supporting customers worldwide.

Kevin Harrison, Client Services Consultant
Kevin Harrison has extensive experience with Db2 for z/OS and System z. He specializes in the performance and tuning of Db2 subsystems, database designs, and application architecture. He has worked with hundreds of Db2 customers, tuning and troubleshooting their applications, and functioning as their Db2 technical advocate.

For more information, go to www.broadcom.com/products/mainframe.

BROADCOM®
MAINFRAME SOFTWARE

For more product information: broadcom.com