IBM zEnterprise Data Compression (zEDC)

Implementation & Exploitation Use Cases

Introduction | Study | Implementation Approach | Use Cases

Authors:

Mai Zeng

Marcelo Lopes de Moraes

Paul W Novak

Pearlson Christopher

Ravinder Akula

Vijayakumar Yeso

Chapter 1 - zEDC Introduction

1.1 - Introduction

Data compression is the process of modifying, encoding or converting the bits structure of data in such a way that it consumes less space on disk. It enables reducing the storage size of one or more data instances or elements. Data compression is also known as source coding or bit-rate reduction.

Due to explosive growth in data, the digital information that exists in the present time will grow from 3.2 zettabytes to 40 zettabytes by 2020. Data is the lifeblood of every organization. It doesn't matter how large or how small an organization is today, data and the ability to access growing volumes of data is critical to running the business.

- I/O throughput is struggling to keep up with increasingly data driven applications.
- Batch workloads are accessing more data from disk and network connections.
- Business opportunities can be lost due to the cost prohibitive nature of keeping data online.

Compression solves problems in the enterprise as it:

- ✓ Improves the effective throughput of data over storage and communication networks
- ✓ Allows more data to remain online for increased business value
- ✓ Reduces the amount of data for encryption operations
- ✓ Typically improves batch turnaround
- ✓ Make storage technology including Flash Memory more affordable

1.2 - Compression implementation on zSystems - Evolution

Informatics -Shrink (1970-1980) - The first commercial product named Shrink, sold in the late 1970s and early 1980s. It used host cycles to perform compression, could generally get about a 2:1 reduction in file size. Sharing data compressed in this manner required accessing the data with the same software that compressed the data to expand it.

Hierarchical Storage Manager (HSM) - This product implemented a storage hierarchy for active/inactive data, and as it moved data for inactive storage, it implemented compression via a Huffman frequency Encoding Algorithm. This algorithm was optimized for writing data so it was an excellent choice for archive and backup data. Only HSM wrote and read these files, so the implementation of this compression did not suffer any data sharing or compatibility issues.

Improved Data Recording Capability (1986) - IBM added a hardware-based data compression option to 3480 Tape called Improved Data Recording Capability (IDRC). This option doubled the capacity of the cartridge (from 200MB to 400MB) and of course required the IDRC feature to be installed on any tape drive that was expected to read or write an IDRC compacted tape volume. IDRC has been a feature of all tape products since its introduction.

Hardware Assisted Data Compression (1994) - In 1994, IBM announced a specific offering on the ESA/390 platform to perform Hardware Assisted Data Compression (HADC) also called Compression Coprocessor (CMPSC). This announcement encompassed a specific set of optimized ES/9000 hardware instructions designed to perform a generic compression of data using the Lempel-Ziv-Huffman algorithm.

RAMAC Virtual Array (1996) - In 1996 the STK Iceberg and later the RAMAC Virtual Array provided a DISK product featuring a Log Structured Array with externalLZ1 compression. The z/OS Host didn't know the data was compressed and this device added the complexity of monitoring the NetCapacity Load (NCL) value to ensure free space existed in the DSS for new data to be added.

zEDC (July 2013) - IBM zEnterprise Data Compression (zEDC) offers a compression acceleration solution designed for high performance, low latency compression with few processor cycles and little additional overhead.

zEnterprise Data Compression (zEDC) Express is an optional native PCIe feature that is available in the z14. It provides hardware-based acceleration for data compression and decompression for the enterprise, which can help to improve cross platform data exchange, reduce CPU consumption, and save disk space.

zEDC is especially optimized for larger data files. z/OS will intelligently determine the larger files which are candidates and directs those to zEDC. zEDC is 'dictionary-less' and uses optimized algorithms to deliver high performance.

IBM Integrated Accelerator for zEnterprise Data Compression (September 2019) - With z15, IBM Integrated Accelerator for zEnterprise Data Compression is provided on each processor chip and uses industry-standard compression formats for file compression that can enable reduction in the size of data which can save storage space and increase data transfer rates. It can also reduce CPU consumption and costs associated with moving, processing, encrypting/decrypting, and otherwise manipulating smaller amounts of compressed data.

IBM Integrated Accelerator for zEnterprise Data Compression interoperates compatibly with the zEDC compression used on previous IBM Z platforms and with industry-standard compression used on other platforms.

1.3 - IBM Z hardware and OS requirements

zEDC requires the following:

- z/OS V2R1 (or later) operating system.
- IBM zEnterprise EC12 CPC (with GA2 level microcode) or zBC12 CPC, or later.
- On z14, zEDC Express feature is sharable across up to 31 partitions and up to 16 cards per CPC.
- zEDC Express software feature enabled in an IFAPRDxx parmlib member.
- Adequate 64-bit real storage configured to this z/OS image.

Chapter 2 - zEDC Study

2.0 - Study Overview

To take advantage of zEDC and promote all the benefits it can deliver, it is important to identify the datasets that are zEDC compression candidates. In this session we are going to analyze system data and discuss how zEDC may impact system processing reading to job elapse time and CPU time utilization.

IBM recommends the use of z Batch Network Analyzer (zBNA) in study phase of zEDC implementation. The good analysis of zBNA reports can help you to determine what workload are most sensitive to changes, improving effectiveness of zEDC implementation.

zBNA needs System Management facility (SMF) data dumped from your systems. It is obtained from CP3KEXTR program which is better described on next topics.

2.1 - zEDC Study Checklist

In this chapter we are going to list all tasks needed for good study and assessment of potential zEDC benefits in your environment:

1. Download and install CP3KEXTR
2. Download and install zBNA
3. Check SMFPRMxx member in z/OS (make sure it is not excluding required SMF types)
4. Make sure your SMF Dump job is not excluding any record type
5. Update and Run EXTR**ALL** / EXTRZ**BNA** members as required
6. Save EDF with .edf extension.
7. Save DATA with .dat extension
8. Save Tersed file with .trs extension

2.2 - CP3KEXTR Download and Installation

CP3KEXTR is used as SMF data extractor for different tools, such as: zBNA, zPCR, and zCP3000. It is offered as a no initial charge application and can be obtained from IBM pages. The CP3KEXTR tool has to be run inside the IBM z/OS system, and the output has to be transferred in text format using File Transfer Protocol (FTP) to the PC where the zBNA tool is installed. The tool is available from the following link:

CP3KEXTR: http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS4229

2.3 - zBNA Download and Installation

zBNA is a PC-based no-charge tool, used to analyze a single batch window of a user-defined length. It collects information from SMF records, analyzes how processors are used and projects how your systems may benefit from zEDC implementation.

You can obtain zBNA from the following links:

IBM clients: http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS5132

Partners: https://www.ibm.com/partnerworld/wps/servlet/mem/ContentHandler/tech_PRS5133

IBM employees: http://w3.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS5126

2.4 - zEDC Study Pre Work

zEDC utilized SMF Types 14 and 15 to create useful reports. However zBNA accepts SMF Types 14, 15, 30, 42, 70, and 113 that must be extracted using CP3KEXTR program. CP3KEXTR is focused on batch jobs, and ignores records that are not batch related.

Before any work with CP3KEXTR or zBNA, make sure that SMFPRMxx member in you system parmlib, and your SMF dump jobs are not is not excluding any important SMF Type record as described above.

After downloading and installing CP3KEXTR and zBNA software, you will be able extract fields from the specific SMF record types and format the data into a flat text file for use by zBNA. It produces an Enterprise Data File (EDF) and a DATA file that are read and used as input to zBNA.

The Enterprise Data File (EDF) is the primary output for zBNA. It is a flat text file created by CP3KEXTR containing summarized data extracted from the SMF scan.

The DATA file is the output file specified by the DATA001 DD card in CP3KEXTR, and can include fields from type 30, 42, 14, and 15 records. It can easily be bigger than 1 GB in large systems, and this can result in long download times, especially if you are remote to the z/OS system that the files are being downloaded from. To reduce impact in this type of situation, zBNA can also support tersed DATA files.

2.4.1 - CP3KEXTR Setup

When you have CP3KEXTR installed, a PDS file (CPSTOOLS.JCL) will be created with sample JCLs.
The CPSTOOLS.JCL dataset has four members already set up for different tools. For zEDC analysis you will use zBNA and only need to use one of the two appropriate member as described below:

Member	User for	Description
EXTR**ALL**	zPCR, **zBNA**, zCP300	Extracts SMF 14, 15 30, 42, 7x, 89, and 113 records into file(s) that can be loaded into PC for zPCR, zBNA or zCP3000. Includes all options, although some less used may appear as comments.
EXTRZ**BNA**	**zBNA**	This is simplified JCL for running the Extract to create all of the data that is used by zBNA (z Batch network Analyzer).

All files mentioned in session 2.4-Study Pre Work are referenced in the CP3KEXTR JCL in following DD cards:

SMFIN: SMF Datasets with your dumped SMF Records
EDF001: Your output EDF file
DATA001: Your output DATA file

2.4.2 - zBNA Setup

After extracting EDF (EDF001) and DATA (DATA001) datasets from CP3KEXTR, you will need to FTP them to you PC and save them as .edf and .dat.

If you prefer to terse your DATA file it must be transferred as binary to the PC, and saved as .trs to be used as input to zBNA.

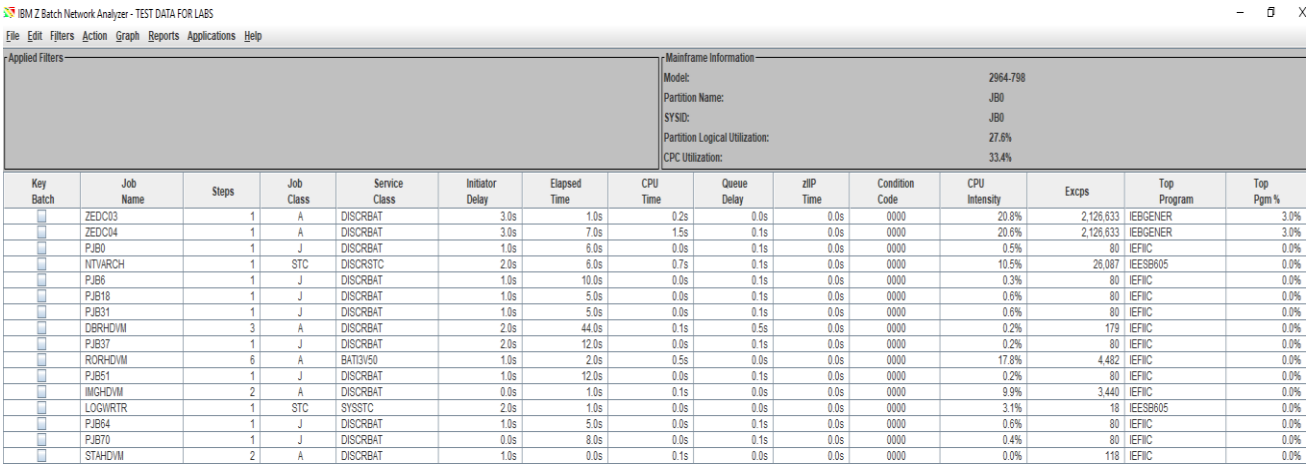
After loading files, you will obtain list of jobs and batch processing metrics found on SMF data loaded. Some of the relevant information for zEDC found in this list are Job Elapsed Time, CPU Time, Queue Delay.

2.5 - zEDC Study

Based on system SMF records, zBNA can be used to identify jobs data sets (BSAM and QSAM) that are zEDC Express compression candidates, across a specified time window (typically a batch window).

Image 2.1 is an example of zBNA result after loading EDF and DATA files:

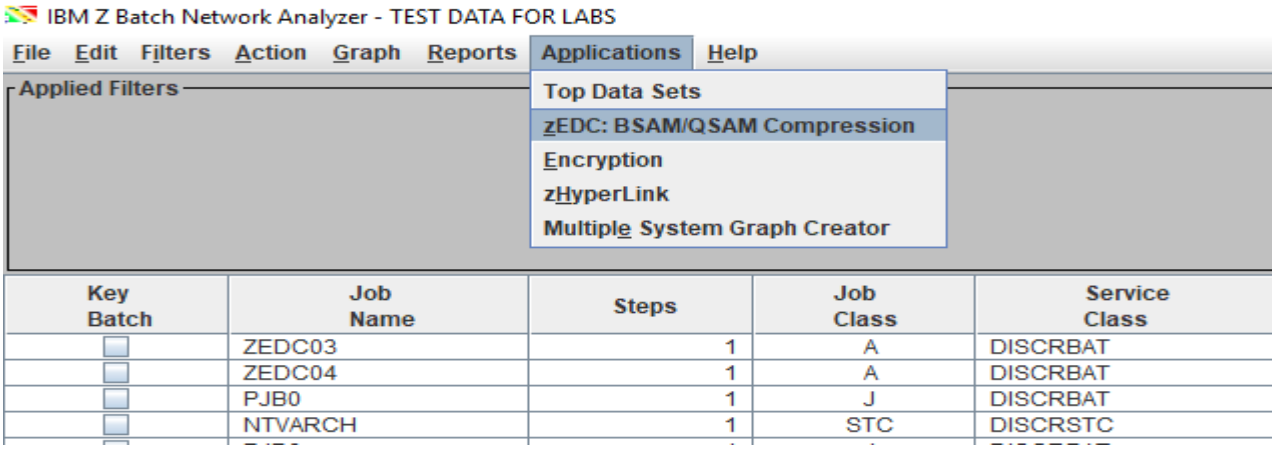
Image2.1



Key Batch	Job Name	Steps	Job Class	Service Class	Initiator Delay	Elapsed Time	CPU Time	Queue Delay	zIIP Time	Condition Code	CPU Intensity	Excps	Top Program	Top Pgm %
	ZEDC03	1	A	DISCRBAT	3.0s	1.0s	0.2s	0.0s	0.0s	0000	20.8%	2,126,633	IEBGENER	3.0%
	ZEDC04	1	A	DISCRBAT	3.0s	7.0s	1.5s	0.1s	0.0s	0000	20.6%	2,126,633	IEBGENER	3.0%
	PJB0	1	J	DISCRBAT	1.0s	6.0s	0.0s	0.1s	0.0s	0000	0.5%	80	IEFIIC	0.0%
	NTVARCH	1	STC	DISCRSTC	2.0s	6.0s	0.7s	0.1s	0.0s	0000	10.5%	26,087	IEESB605	0.0%
	PJB6	1	J	DISCRBAT	1.0s	10.0s	0.0s	0.1s	0.0s	0000	0.3%	80	IEFIIC	0.0%
	PJB18	1	J	DISCRBAT	1.0s	5.0s	0.0s	0.1s	0.0s	0000	0.6%	80	IEFIIC	0.0%
	PJB31	1	J	DISCRBAT	1.0s	5.0s	0.0s	0.1s	0.0s	0000	0.6%	80	IEFIIC	0.0%
	DBRHDVM	3	A	DISCRBAT	2.0s	44.0s	0.1s	0.5s	0.0s	0000	0.2%	179	IEFIIC	0.0%
	PJB37	1	J	DISCRBAT	2.0s	12.0s	0.0s	0.1s	0.0s	0000	0.2%	80	IEFIIC	0.0%
	RORHDVM	6	A	BATI3V50	1.0s	2.0s	0.5s	0.0s	0.0s	0000	17.8%	4,482	IEFIIC	0.0%
	PJB51	1	J	DISCRBAT	1.0s	12.0s	0.0s	0.1s	0.0s	0000	0.2%	80	IEFIIC	0.0%
	IMGHDVM	2	A	DISCRBAT	0.0s	1.0s	0.1s	0.0s	0.0s	0000	9.9%	3,440	IEFIIC	0.0%
	LOGWRTR	1	STC	SYSSTC	2.0s	1.0s	0.0s	0.0s	0.0s	0000	3.1%	18	IEESB605	0.0%
	PJB64	1	J	DISCRBAT	1.0s	5.0s	0.0s	0.1s	0.0s	0000	0.6%	80	IEFIIC	0.0%
	PJB70	1	J	DISCRBAT	0.0s	8.0s	0.0s	0.1s	0.0s	0000	0.4%	80	IEFIIC	0.0%
	STAHDVM	2	A	DISCRBAT	1.0s	0.0s	0.1s	0.0s	0.0s	0000	0.0%	118	IEFIIC	0.0%

Under Applications option you will find different application reports, and we are going to work with zEDC reports. See image 2.2

Image 2.2



Key Batch	Job Name	Steps	Job Class	Service Class
	ZEDC03	1	A	DISCRBAT
	ZEDC04	1	A	DISCRBAT
	PJB0	1	J	DISCRBAT
	NTVARCH	1	STC	DISCRSTC

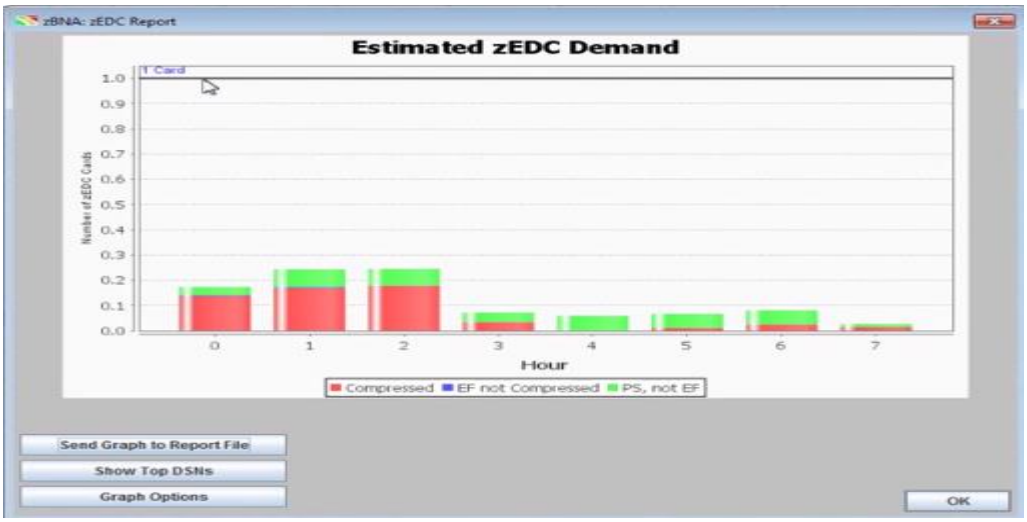
*Some zBNA versions may present different options on Main Menu. All reports used here were collected from latest version available in May 2019 (zBNA 1.8.2)

With the zEDC reports you will find a list of data sets by jobs:

- Jobs that already perform hardware compression and might be candidates for zEDC Express
- Jobs that might be zEDC Express candidates, but are not in extended format

Furthermore, with these reports can estimate the use of a zEDC Express feature, and the number of features needed, as showed on **image 2.3.**

Image 2.3



In this graphical report you may have a better idea about number zEDC features required in your environment. This analysis is done based on what is found on SMF records during the batch window extracted by CP3KEXTR. Note in the example above, that 1 zEDC Card would be enough in this environment.

Besides the number of features, you have additional information about your environment, such as amount of zEDC candidate datasets categorized by:

- Datasets being compressed by traditional compression
- Extended Format Datasets not compressed
- Physical Sequential Datasets (Not Extended Format)

2.6 - Analyzing zEDC Reports

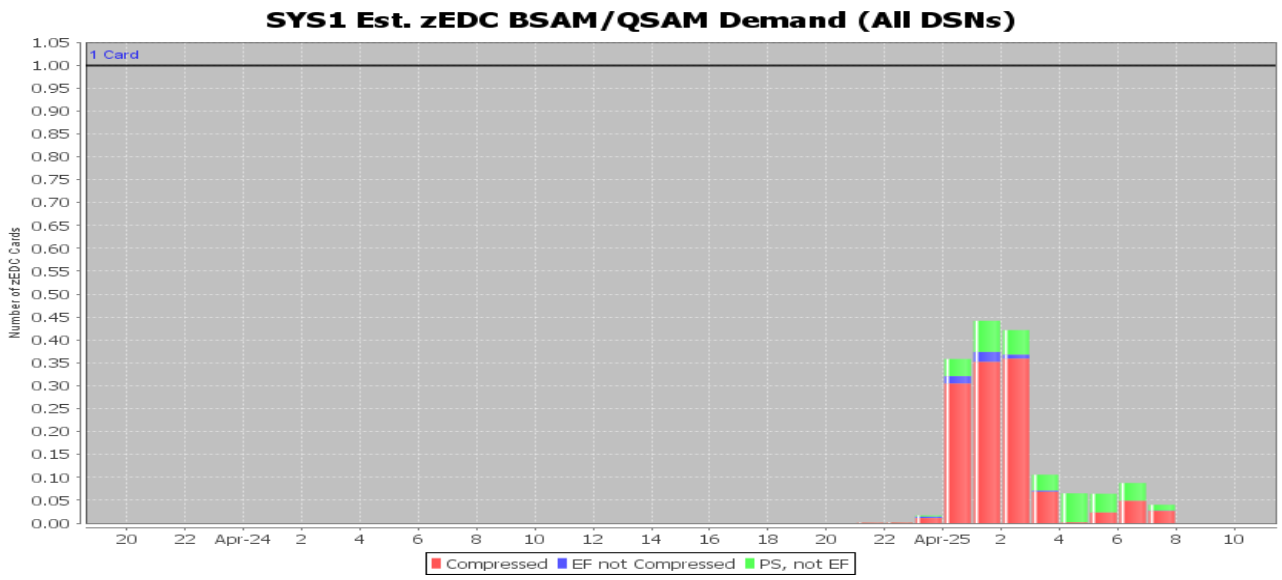
Different variables need to be considered and studied carefully before zEDC implementation, this may avoid surprises and help to reach expected results. We are going to discuss further each of the important variables from zEDC reports

2.6.1 Number of zEDC Cards

In this report we have number of cards needed for the environment being studied, and the amount of zEDC Card utilization, illustrated and categorized in 3 types of data: Compressed, Extended Format Compressed, and Physical, not Extended Format.

Estimated zEDC Cards

Image 2.4



Number of Cards – Gives you the number of zEDC features required. In the example above, 1 card would be enough if it is the only system running in the CEC.

Compressed – Those are datasets being compressed by traditional compression. In this scenario it is important to consider that CP processing is needed to complete traditional compression, so there is an opportunity here to explore zEDC. If zEDC is implemented, you will potentially have significant CPU processing back and it would be processed by zEDC as illustrated on image 2.4. These datasets may be where you should invest most of your time with the customer, once it represents the highest zEDC utilization as showed above.

Extended Format not Compressed – The entries collected in this metric, are the files defined as Extended Format but are not being compressed. With zEDC implementation a little more CPU may be consumed during compression of these datasets, but you may get significant pure I/Os and the batch processes may get shorter elapsed time, which would indicate less CPU time to run the batch and less storage consumption. It should be better looked to determine that.

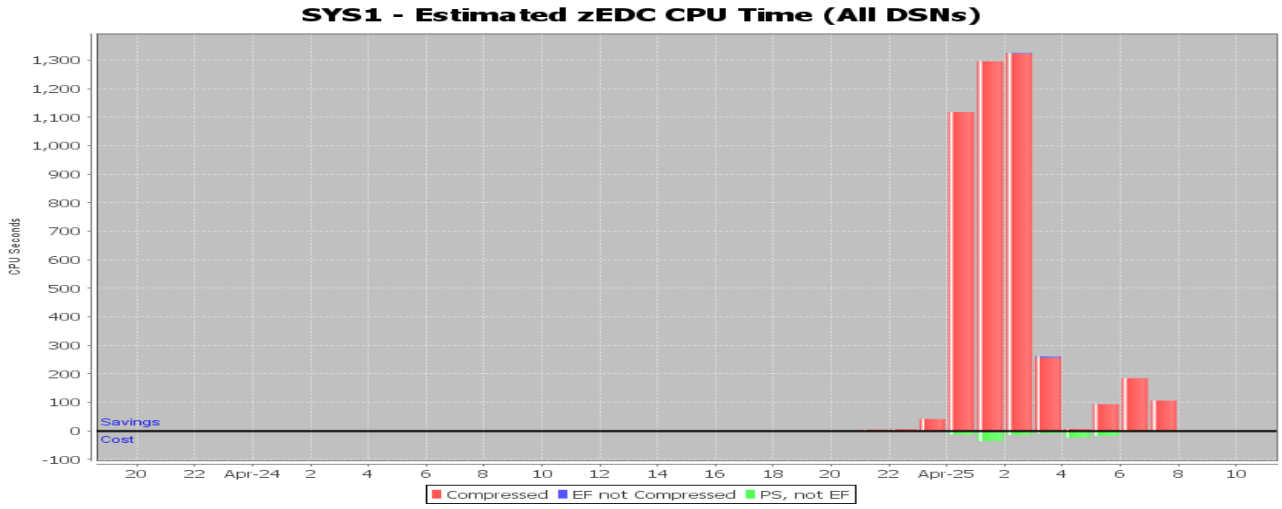
Physical Sequential not Extended Format – These are datasets not being compressed and they are not defined as Extended Format. They are zEDC candidates but need to be converted to Extended Format. It means additional work to the customer involving ACS routines and other SMS system settings to make those files ready to be compressed. Depending on amount of PS datasets and average I/O time taken to Read/Write these datasets you may have considerable CPU saving with zEDC and reduce amount of storage needed in your environment. It also need to be better looked to determine a gain after zEDC implementation

2.6.2 - CPU Time

In the CPU Saving Report you can identify and show customers how much of CPU processing may be reduced with the use of zEDC. It is also categorized by type of data (Compressed, EF not Compressed, PS not EF).

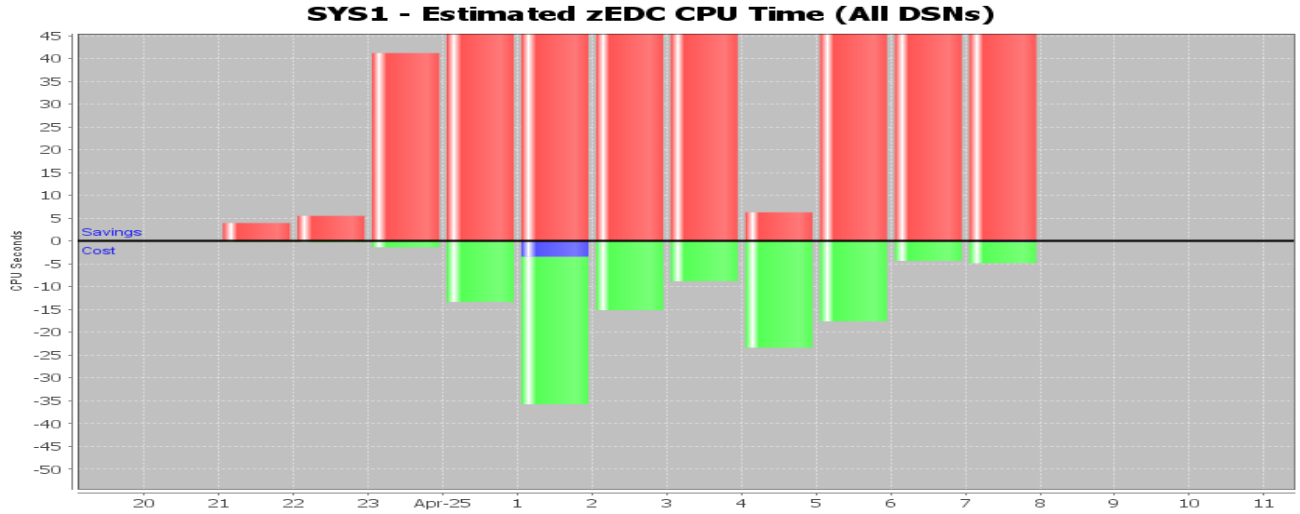
Estimated zEDC CPU Saving

Image 2.5



Note that compressed files are the greater contributors for CPU saving. Here you see how much of CPU Time is saved on may be reduced with zEDC implementation per timeframe

Image 2.6



The image 2.6 has the same data with a zoom on specific areas of the graph.

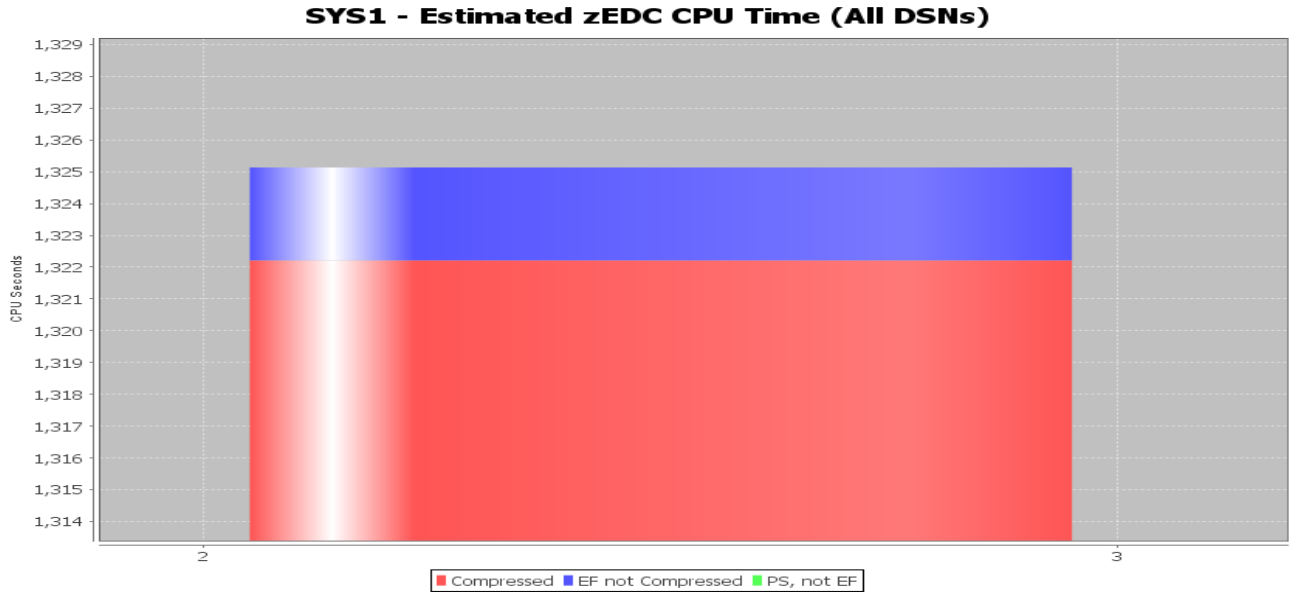
In this scenario, it is clear that at from CPU perspective, there is would be no gain on using zEDC over Physical Sequential not Extended datasets. There is no CPU time saving, only additional costs, once additional CPU time would be required to compress these files with zEDC. It is a small portion of the CPU time, and in a very specific time frame, but we must consider all the additional work required to convert those files to Extended Format too.

This is not the only determinant metric to define if it would be worthy to convert these datasets to zEDC compression or not, other studies need to be done before a final decision, however, this is one of the most important information to be considered since CPU time saving is of the main reasons for implementing zEDC.

A similar scenario may be noticed with Extended Files not Compressed. They would demand less CPU time compared to PS not Extended files, but that might be because of the smaller number of datasets processed during this time frame. There were 90 EF not Compressed datasets versus 2227 PS not Extended datasets.

The Image 2.7 has more details about EF not compressed files. It shows that, there is a small gain on CPU time from 2am to 3am, but it shouldn't be relevant in this scenario

Image 2.7



2.6.3 - CPU MIPS and MSU

The data presented here in the CPU analysis shows a very good potential to explore zEDC. Traditionally compressed data are clearly the main candidates for zEDC, and as covered in previous report concerned to CPU, here we also note a considerable saving on MIPS, following the same proportion of CPU Time savings.

Datasets compressed by traditional compression methods should be considered and used in a zEDC discussion with the customer

Estimated zEDC MIPS

Image 2.8

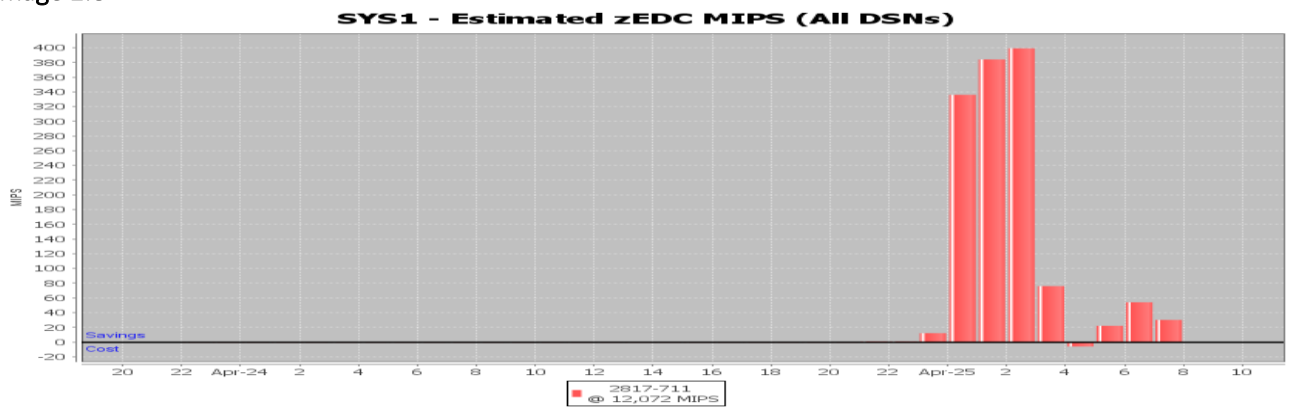
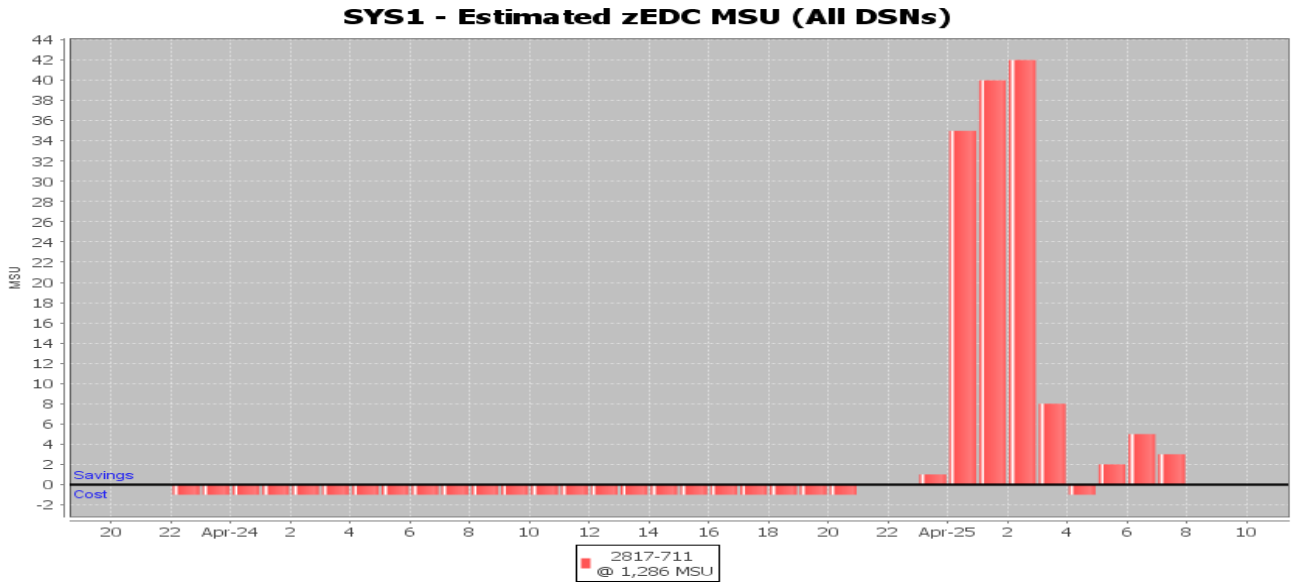


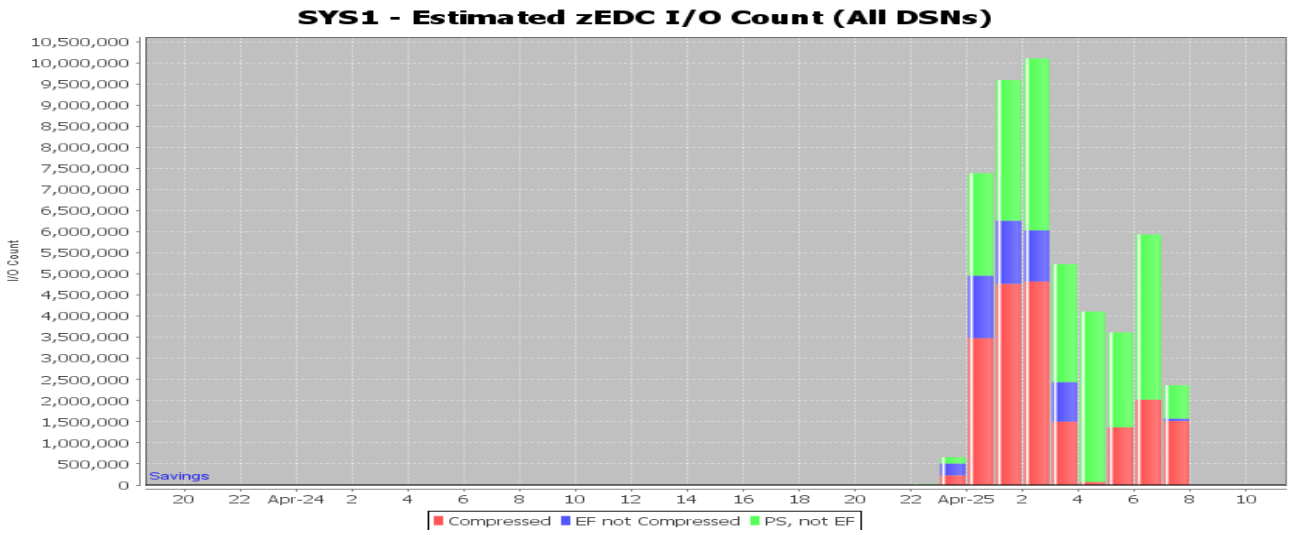
Image 2.9



2.6.4 - I/O Count

The I/O count represented here mirrors the trending of compressed datasets from previous graphs, but indicates different conclusion about zEDC for EF not Compressed files and Physical Sequential not Extended Files.

Image 2.10



You could see on previous reports that the only scenario representing CPU saving after zEDC deployment is for compressed datasets, however, CPU statistic is not the only variable to be analyzed.

The Image 2.10 shows the graph for number of I/Os categorized by type of data, and even not providing any gain over CPU processing, PS not EF datasets and EF not Compressed have high potential to provide great advantages to this system decreasing I/O count significantly after zEDC implementation.

Depending on the balance between I/O count savings and CPU costs caused by zEDC implementation, it may offer great advantages, it would depend on what is in focus for that client. This behavior may be explained by the fact that this system has used EF not Compressed and Ps not EF datasets in I/O bound processes, while traditional compressed files are usually CPU bound processes.

2.7 - Study Conclusion

After analyzing all these reports, this system analysis shows that traditional compressed files are the main reason for implementing zEDC, they offer CPU Time Savings, CPU MIPS and MSU Savings, and reduced I/O count. Physical Sequential datasets were not being benefited with zEDC implementation, they were not providing any CPU improvement, and they were costing more after migrating to zEDC compression. They also need additional work to be converted to Extended Format.

However, the additional CPU seconds needed to process compression of these datasets are not so relevant compared to the gain it offers over I/O Counts. We are not considering workforce to convert those datasets to extended format, and this should be analyzed in conjunction with the customer, but considering CPU vs I/O metrics, we would recommend the conversion of Physical Sequential datasets to zEDC cards.

Extended Format not compressed datasets follows the same logic of Physical Sequential datasets, they require even less CPU seconds after zEDC conversion, less than 5 seconds as showed on Image2.6. This is irrelevant compared to the number of I/O's decreased after zEDC implementation, resulting on reduced batch windows and better storage resources utilization.

Although CPU is generally the most important resource for most of customers, and CPU savings are usually the main reason why customers may want to implement zEDC, it is not the only benefit taken from it. As showed in this study case, zEDC cards are great contributors for different type of processes, I/O bound processes and CPU bound processes.

Chapter 3 - zEDC Implementation Approach

In this chapter, we describe the zEDC implementation requirements at System level and Data-class level.

3.1 - Implementation approach at DFSMS level

There are two methods to apply zEDC, *System level and Dataclas level*.

The preferred method is at Dataclas level as it provides better control over the data allocation.

Step 1: Create a new DATACLAS for zEDC.

Define the Data Class, with the COMPACTION parameter set to ZP and Data set type as EXTENDED format.

Note: zEDC compressed data sets are defined as extended format version 2 data sets, regardless of the user's specification.

Step 2: Modify the ACS Data Class routine to provide specific data set patterns for the zEDC-compressed files.

Note: zBNA identifies BSAM/QSAM data sets that are candidates for zEDC.

Step 3: DFSMSdss usage: Specify the ZCOMPRESS(PREF) parameter in DFDSS SYSIN JCL, the contents of the dump data set are compressed using zEDC compression services.

Step 4: DFSMShsm: ZCOMPRESS is an optional parameter set that you use to specify the type of compression used during migration or backup for all data sets.

Note: ZCOMPRESS parameter is used only for SMS data sets that are not compressed on L0; for non-SMS data sets, it is always in effect, as you cannot have a non-SMS data set compressed on L0.

Configure ARCCMDxx with the following parameters for zEDC compression.

SETSYS ZCOMPRESS(TAPEMIGRATE(YES))

SETSYS ZCOMPRESS(DASDMIGRATE(YES))

SETSYS ZCOMPRESS(TAPEBACKUP(YES))

SETSYS ZCOMPRESS(DASDBACKUP(YES))

zEDC for DFSMShsm volume dumps.

A new DEFINE DUMPCLASS optional parameter is available:

DEFINE DUMPCLASS(ZCOMPRESS(NO | YES))

- In addition to existing Tailored (T) and Generic (G) values, new zEDC Required (ZR) and zEDC Preferred (ZP) values are available on the COMPACTION option in data class.
- When COMPACTION=Y in data class, the system level is used.
- Compaction . . . . . ZP (Y, N, T, G, ZR, ZP or blank)
- zEDC Compressed Format data sets are created as Version 2 data sets regardless of the user's specification in DataClass, JCL or SYS1.PARMLIB

3.2 - Implementation approach at System level

System level In addition to existing TAILORED and GENERIC values, new zEDC Required (ZEDC_R) and zEDC Preferred (ZEDC_P) values are available on the COMPRESS parameter found in IGDSMSxx member of SYS1.PARMLIB.

COMPRESS(TAILORED|GENERIC|zEDC_R|zEDC_P)

Activate using SET SMS=xx or at IPL

Data class continues to take precedence over system level.

Note - when the default system level is modified to specify zEDC_R or zEDC_P, all new allocation requests using a data class with COMPACTION=Y will request zEDC compression.

New Compaction values:

ZP: Prefer zEDC compression - The system will not fail the allocation request but rather create either a tailored compressed data set if the zEDC function is not supported by the system or create a non-compressed extended format data set if the minimum allocation amount requirement (5MB,or 8MB Primary if no Secondary) is not met.

ZR: Require zEDC compression - The system will fail the allocation request if the zEDC function is not supported by the system or the minimum allocation amount requirement (5MB, or 8MB Primary if no Secondary) is not met.

3.3 - Other implementation considerations

For systems that do not support the zEDC Express feature, but have z/OS V2.1, z/OS V1.13, or z/OS V1.12 installed, it is possible to access a zEDC Express compressed-format data set. In this case, compressed data is read from data sets and decompressed using software algorithms. New data being written is not compressed.

Note: Software decompression is slow, and uses considerable processor resources. Therefore, it is not suggested for production environments.

Existing QSAM/BSAM data sets (whether compressed or not) must be copied to a new target data set allocated with zEDC compression. No utility available to perform a conversion without de-compressing source and recompressing target. Normal tools can be used to perform the copy, for example IEBGENER or REPRO

3.4 - Implementation best practices

- zEDC compression at DATACLAS level rather than system level. Use zBNA to identify eligible data sets for zEDC compression.
- Recommended not to implement the zEDC compression globally across all sequential files. Rather break it down into several parts and implement methodically.
- GDG / GDS would be the good candidates for the zEDC since they are sequential.
- DB2 Archive: Implement the zEDC compression for ARCH log 2/COPY2 before implementing on all ARCHLOGS.

- Create a test scenario for all type of eligible data before implementation.
- Manage zEDC compression using ACS routines rather than using at task level (DFSMSdss / SVC dumps etc).

3.5 - Few problem scenarios and solutions

Problem #1: Allocation to DC zEDC caused "IEC141I 013-4C“ abend for some data sets.

Cause: The problem was caused by the Data class parm "RECORD ACCESS BIAS=USER" . .

A value of USER prevents SYSTEM MANAGED BUFFERING and number of buffers. Buffering algorithms will be based on user specified (or defaulted) values and current algorithms. The data sets which encountered this problem were earlier using a Dataclas that doesn't use RECORD ACCESS BIAS. The data sets that failed had buffer pools created based on what was coded in the JCL or default (and not system determined), which messed up the jobs.

Solution: Split the zEDC dataclas based on the data sets being pushed for zEDC. The ones that need system determined buffer size etc, need to have "RECORD ACCESS BIAS = -----".

Problem #2: ZEDC compression was assigned to smaller data sets.

Cause: The problem was caused by the ACS routine variables used &MAXSIZE.

Since the &MAXSIZE considers an estimate of the size a data set can grow up to (P + 15xS), it caused data sets which were smaller but allocated with large extent size to, pick ZEDC Dataclas.

Solution: Use a combination of &SIZE and &MAXSIZE to select data sets eligible for ZEDC Dataclas, may not prove 100% effective though.

Consider an example - (&SIZE > 150 MB & &MAXSIZE > 500MB) would allow data sets which have Primary extent size larger than 150 MB and expected total allocated size of larger than 500 MB, to pick ZEDC DC. That means a data set would need to have the PRIMARY extent size > 150 MB and Secondary extent size > 24 MB, in order to be eligible (even if it hasn't allocated all the extents yet).

```
WHEN ((&DSTYPE = 'GDS') AND (&DSN EQ &ZEDC_FILTER1) AND
    (&SIZE GE 200000kb) AND (&MAXSIZE GE 500000KB) AND (&PGM ¬= ‘NONZEDC'))
  DO
   SET &DATACLAS = 'DCZEDC'
   EXIT CODE(0)
  END
```

Chapter 4 - zEDC Use Cases

4.0 - zEDC Exploiters overview

- SMF logstreams
  For increased availability and online storage reduction.
- QSAM and BSAM (sequential) data sets
  For better disk use and batch elapsed time improvements.
- DFSMSdss/DFSMShsm
  When backing up and restoring data, or migrating and recalling data.
- IBM Java V7.0.0 SR7 and Java V7R1 runtime environment
  For high throughput standard compression with java.util.zip.
- IBM Encryption Facility for z/OS V2.1
  For building industry-standard compressed OpenPGP1 (RFC4880) files.
- IBM Sterling Connect: Direct for z/OS V5.2
  For better throughput and link use.
- IBM WebSphere® MQ for z/OS V8 and later
  For channel message compression.
- zlib
  For application programs that directly use the zEDC with the zlib open source library application programming interfaces (APIs).

4.1 - zEDC for MQ Channel Message Compression

WebSphere MQSeries has always provided compression options for message data passed over MQ channels via the COMPMSG attribute. The existing zlib options are the following:

- ZLIBFAST - Message data compression is performed using the zlib compression technique. A fast compression time is preferred.
- ZLIBHIGH - Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

Starting with WebSphere MQ for z/OS V8 the COMPMSG(ZLIBFAST) attribute will now use zEDC when available to perform compression and decompression of message data.

- Scenario description: zEDC Compress for MQ Channel Message
- Test Environment:
  - 2 Parallel SYSPLEXes with z/OS V2R3
  - Qrep V1140

- Db2z V12
- MQ 9.1

- Test Method:
  - Benchmark :
    - MQ Channel : COMPMSG NONE ---no compression for message
    - Trigger simulated OLTP and Batch workload, and check performance
  - zEDC compress for MQ Channel Message
    - Set MQ Channel into COMPMSG(ZLIBFAST)
    - Check the zEDCs which are in ALLC status.
    - Recycle the related channels
    - Trigger the same simulated OLTP and Batch workload, and check performance.
  - Preload scenarios
    - Based on the existing Db2 log, run the preload testing for both OLTP and Batch with COMPMSG NONE and ZLIBFAST.

In Qrep start proclib, specify the earlier LSN as the workload beginning time to recycle QCapture.

- Test Result: When the packaged message size less than 4 KB(in the OLTP time), zEDC compression couldn't take effect. But when the size is more than 4 KB(in the batch time), such as 50KB or above, the compression ratio can achieve 1:5. So for QCAP parameter, TRANS_BATCH_SZ can be set a big value such as 16 if there aren't too many key-dependences. It's definitely useful to relief the bandwidth pressure.

- Feature Usage Scenario:
  - Active-Active solution with insufficient network bandwidth

4.2 - zEDC for SMF logger

- Scenario description:
  - zEDC Compress with z/OS SMF Logger which will reduce the amount of storage
- Test Environment:
  - z/OS V2R3 where SMF is in logstream mode and COMPRESS attribute
- Test Method:
  - Create two log streams with the same SMF records type, one is for non-compression and the other one for compression:

- LSNAME(IFASMF.&SYSNAME..NOZEDC,TYPE(30,70:79,100:102)) -- non compression mode
- LSNAME(IFASMF.&SYSNAME..ZEDC,TYPE(130,70:79,100:102),COMPRESS(PERMFIX(1024M)))

  - Collect 10 minutes SMF data, then compare the bytes written for non-compressed and compressed by SMF88 record.

- Test Result:

  - LOGSTREAM IFASMF.JA01.NOZEDC: non-compress data before getting into log stream, the amount of bytes written is about 2 GB. LOGSTREAM IFASMF.JA01.ZEDC: compressed data before entering log stream, the amount of bytes written is about 149MB. So the storage amount of compress log stream is only 7.28% comparing with non-compressed log stream.

4.3 - zEDC for BSAM and QSAM Data Set

- Scenario description:

  - zEDC Compress for BSAM & QSAM data set will reduce the amount of storage

- Test Environment:

  - z/OS V2R3 and data set are in Extended type

- Test Method:

  - A. For the sequential data set, use IEBGENER as access method to read and write the files we created with BSAM.
  - B. Continue to read/write the same files we created before using IEBDG and QSAM
  - C. After the test run, we compared the input and the output data sets. By looking at the output data set, we confirmed that DFSMS and the zEDC Express feature where properly configured.
  - Test result: The newly created output data set was allocated as an extended-format sequential data set and the output data set is compressed to about 1:10 ratio

```
Data Set Name . . . . : ZENGMAI.TEST.D190802.NOZEDC

General Data                            Current Allocation
 Management class . . : **None**         Allocated cylinders : 250
 Storage class  . . . : **None**         Allocated extents . : 5
  Volume serial . . . : SP0009
  Device type . . . . : 3390
 Data class . . . . . : **None**
  Organization  . . . : PS               Current Utilization
  Record format . . . : FBA               Used cylinders  . . : 226
  Record length . . . : 1024              Used extents  . . . : 5
  Block size  . . . . : 10240
  1st extent cylinders: 50
  Secondary cylinders : 50              Dates
  Data set name type  :                  Creation date . . . : 2019/08/02
  Data set encryption : NO               Referenced date . . : 2019/08/02
                                         Expiration date . . : ***None***

 SMS Compressible  . : NO
```

```
Data Set Name . . . . : ZENGMAI.ZEDC.D190802.BSAM

General Data                            Current Allocation
 Management class . . : STANDARD         Allocated cylinders : 14
 Storage class  . . . : STANDARD         Allocated extents . : 1
  Volume serial . . . : TSO01B
  Device type . . . . : 3390
 Data class . . . . . : ZEDC            Current Utilization
  Organization  . . . : PS               Used cylinders  . . : 14
  Record format . . . : FBA              Used extents  . . . : 1
  Record length . . . : 1024
  Block size  . . . . : 10240
  1st extent cylinders: 14
  Secondary cylinders : 50              Dates
  Data set name type  : EXTENDED         Creation date . . . : 2019/08/02
  Data set encryption : NO               Referenced date . . : 2019/08/02
                                         Expiration date . . : ***None***

SMS Compressible  . : YES
```

- ZEDC data class:

```
Data Set Name Type  . . . . . : EXTENDED
  If Extended . . . . . : REQUIRED
  Extended Addressability . . : YES
  Record Access Bias  . . . . : USER
  RMODE31 . . . . . . . . . . :
Space Constraint Relief . . . : NO
  Reduce Space Up To (%)  . . :
  Guaranteed Space Reduction  : NO
  Dynamic Volume Count  . . . :

Compaction  . . . . . : ZR
```

4.4 - zEDC Compression with Java

- Scenario description:
  - Make use of java.util.zip classes to compress the Java with high throughput
- Test Environment:
  - z/OS V2R3 with java version "1.8.0_161" Java(TM) SE Runtime Environment (build 8.0.5.10 (SR5 FP10)) and IBM J9 VM (build 2.9, JRE 1.8.0 z/OS s390x-64 Compressed (JIT enabled, AOT enabled)

- **Test Method**: java FileCompressionUtil -zip joblog_08_16_2019.zip /AT21/var/log
  - A. A directory with many files as the input
  - B. Set the Java application to compress files using the java.util.zip classes
  - C. Check the output size of zip file
- **Test result:**
  - The ZIPed package size is only about 6.78% of the original directory.

```
131:/u/zm $ du /AT21/var/log
  905888 /AT21/var/log

147:/u/zm $ du joblog_08_16_2019.zip
   61376 joblog_08_16_2019.zip
```
