Compression on z Systems

Jan Tits
IBM Belgium
jantits@be.ibm.com
9 December 2015
Agenda

- Why compression is important
- Compression on z Systems
- z13 CPU Compression Improvements
- What is zEDC?
- zEDC Use Cases
- Application Programming with zEDC
- Usage Monitoring
Explosive growth in data
Every day over 2000 petabytes of data are created

Data Compression will become pervasive
- 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

Data needs to be shared across different platforms
- Data is being exchanged among business partners
- Compression can substantially reduce the amount of data transferred
- Industry standard formats need to be used for transparent peer to peer communication

Compression solves problems in the enterprise
- 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

Storage technology, including Flash, more affordable with compression
**z Systems Compression Technology Overview**

**Using the right hardware compression acceleration for each of your workloads**

### Use Cases

<table>
<thead>
<tr>
<th>Small object compression</th>
<th>Large Sequential Data</th>
<th>Industry Standard Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows in a database</td>
<td>QSAM/BSAM Online Sequential Data</td>
<td>Cross Platform Data Exchange</td>
</tr>
<tr>
<td>Objects stored in a database</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Users

- **Small object compression**
  - VSAM for better disk utilization
  - DB2 for lower memory usage
  - The majority of customers are currently compressing their DB2 rows

- **Large Sequential Data**
  - QSAM/BSAM for better disk utilization and batch elapsed time improvements
  - SMF for increased availability and online storage reduction

- **Industry Standard Data**
  - Java for high throughput standard compression via java.util.zip
  - Encryption Facility for z/OS for better industry data exchange
  - IBM Sterling Connect: Direct® for z/OS for better throughput and link utilization
  - ISV support for increased client value
Benefits of Faster CPU Compression with IBM z13

The z13’s compression engine reduces CPU needed for compression with CMPSC up to 50% compared to the zEC12. This can provide significant CPU performance and response time advantages for applications and middleware using compression.

Our measurements have shown that I/O intensive batch jobs using QSAM/BSAM Generic and Tailored compression can take up to 49% less CPU and up to 40% less elapsed time on z13 than on zEC12.

Disclaimer: Measurement results were collected in a controlled measurement environment. Your results may vary.
The advantages of compressing DB2 tables are well understood. DB2 compression can significantly reduce the DASD space needed to store data. DB2 compression can also result in improved buffer pool hit ratios, reducing IO operations and helping response time and CPU consumption. DB2 gets significant benefit from improved CPU compression performance on the z13.

Loading data to compressed tables, the CPU cost of compression is reduced by up to 59% on z13 as compared to zEC12. Given this improvement, the CPU cost of loading to compressed tables vs. uncompressed tables can be as low as 8% on z13.

Unloading data from compressed tables, the CPU overhead of compression is reduced by up to 69% on z13 as compared to zEC12. Given this improvement, the CPU cost of unloading from compressed tables vs. uncompressed tables can be as low as 14% on z13.

Disclaimer: Measurement results were collected in a controlled measurement environment. Your results may vary.
IBM zEnterprise Data Compression (zEDC)
New data compression offering that can reduce resource usage

**What is it?**
- zEDC Express is an IO adapter that does high performance industry standard compression
- Used by z/OS® Operating System components, IBM Middleware and ISV products
- Applications can use zEDC via industry standard APIs (zlib and Java™)
- Each zEDC Express sharable across 15 LPARs, up to 8 devices per CEC.
- Raw throughput up to 1 GB/s per zEDC Express Hardware Adapter vs typical 50 MB a second in software

**What Changes?**
It is time to revisit your decisions about compression.
- **Disk Savings**: Many people are already getting value from CMPSC compression and software compression today
- **Performance**: High throughput alternative to existing IBM System z® compression for large or active files.
- **Industry Standard**: Low cost compressed data exchange across all platforms
- **Pervasive**: Standard APIs allow quick adoption by middleware products running on System z

**What is the Value?**
New sources of customer value
- **QSAM/BSAM** can save up to 4x disk space and in some cases shorten elapsed time, reducing batch windows.
- **Business Partner Data Exchange** can have higher throughput with lower CPU cost
- **Managed File Transfer** saves up to 4x link bandwidth, and up to 80% elapsed time
- **ISV Products** deliver expanded customer value
- **Java for z/OS V7R1** accelerates common compression classes used by applications and middleware
- **Improved availability** with SMF
zEDC Product Usage Overview

Roadmap of Growing Value and Commitment to Compression

• Value buy-in from IBM products across the product stack
• z Systems ISVs have been very active with zEDC
Sequential Data Set Compression with BSAM/QSAM and zEDC

Reduce the cost of keeping your sequential data online

- zEDC compresses data up to 4X, saving up to 75% of your sequential data disk space
- Capture new business opportunities due to lower cost of keeping data online

Better I/O elapsed time for sequential access

- Potentially run batch workloads faster than either uncompressed or QSAM/BSAM current compression

Sharply lower CPU cost over existing compression

- Enables more pervasive use of compression
- Up to 80% reduced CPU cost compared to tailored and generic compression options

Simple Enablement

- Use a policy to enable the zEDC

Example Use Cases

**SMF Archived Data** can be stored compressed to increase the amount of data kept online up to 4X

**zSecure** output size of Access Monitor and UNLOAD files reduced up to 10X and CKFREEZE files reduced by up to 4X

Up to 5X more **XML** data can be stored in sequential files

**The IBM Employee Directory** was stored in up to 3X less space

**z/OS SVC and Stand Alone DUMPs** can be stored in up to 5X less space

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
QSAM/BSAM zEDC

Uncompressed

Current zEC12 Compression

Data Set Type

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
Data Replication

- Replication technologies which move data in physical format can take advantage of the reduced storage requirements of data compressed with zEDC.
  - Significant amounts of zEDC compressed data can reduce the amount of data transferred as well as the elapsed time to complete the transfer.
Sterling Connect:Direct can automatically leverage zEDC Express Accelerator for file compression and decompression as files are transferred.

Works with various dataset and file types on z/OS.

File transfers can be z/OS to z/OS, or z/OS to Distributed (UNIX®, Linux®, Linux on System z, and Windows).

Fully compatible with zlib compression used in IBM Sterling Connect: Direct today – no changes required at end points.

Up to **80%** reduction in elapsed time to transfer a file from z/OS to z/OS (results vary by dataset type and characteristics of the data) with minimal CPU increase.

Significant improvement in CPU time with zEDC over zlib software compression (sender TCB time and receiver TCB time).

Users currently using compression will see a comparable compression ratio.

Disclaimer: Results based on internal controlled measurements. Results may vary by customer based on individual workload, data, configuration and software levels.
QSAM/BSAM Data Set Compression with zEDC

- Setup is similar to setup for existing types of compression (generic and tailored)
  - It can be selected at either or both the data class level or system level.
    - **Data class level**
      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
    - **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.
  - **Activated using SET SMS=xx or at IPL**
    Data class continues to take precedence over system level. The default continues to be GENERIC.

- **QSAM/BSAM Conversion**
  - Convert QSAM/BSAM data sets into zEDC format by copying data into a new zEDC compressed format data set using standard utilities (IEBGENER, IDCAMS REPRO)

- The new zEDC compression for new extended format data sets is **Optional**
  - All previous compression options are still supported
  - For the full zEDC benefit, zEDC should be active on ALL systems that might access or share compressed format data sets. This eliminates instances where software inflation would be used when zEDC is not available.
HSM/DSS Usage of zEDC

- **DFSMSdss DUMP command**
  - In addition to existing COMPRESS and HWCOMPRESS options on DUMP command, new **ZCOMPRESS (REQUIRED | PREFERRED | NONE)** option will take advantage of zEDC compression.
  - Accepted for all FULL, TRACKS, physical and logical DATASET backups to DASD and tape
  - The use of zEDC for backups can be restricted using a new facility class profile: STGADMIN.ADR.DUMP.ZCOMPRESS

- **DFSMShsm** will use the **DFSMSdss zEDC** support (via **ZCOMPRESS(PREFERRED)**) in
  - Migrate/Recall
  - Backup/Recover
  - Full Volume DUMP
  - Recover and FRRECOV from DUMP

Exception: zEDC will not be used during migration or backup functions when DFSMShsm is the data mover. Partitioned Data Sets will utilize the standard DFSMShsm compaction methodology in place.

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
Benefits using DFSMSHsm Migration/Recall with zEDC
Reduced VTS and Tape Usage

DFSMSHsm

Active Data (ML0)

Migrate to Physical Tape

Migrate to ML2 Tape and gain up to:
11% less CPU*,
22% less space and
30% less elapsed time

Recall from Tape and gain up to:
9% less CPU* and
34% less elapsed time

Fewer cartridges to manage

Migrate to ML2 Virtual tape and gain up to:
15% less CPU*,
37% less space and
25% less elapsed time

Recall from Virtual Tape and gain up to
11% less CPU* and
32% less elapsed time

Less virtual cache space used

Less virtual cache space used

*as compared to using DFSMSHsm with no host compaction options on z13

Results measured in an internal lab environment. Your results will vary.
Benefits using DFSMSHsm with zEDC for ML1 Migrations
Compress data with less CPU

Migrate and use: Up to 58% less space Up to 80% less CPU*

Recall from disk with 69% reduction in CPU*

Side benefit: Shorter elapsed time

*as compared to using DFSMSHsm with the COMPACT option on zEC12.

When migrating data to ML1 Disk using zEDC, use up to 58% less disk space and use up to 80% less CPU compared to using DFSMSHsm with the COMPACT option. When using DFSMSHsm with zEDC, recalling data from ML1 Disk uses up to 69% less CPU as compared to using DFSMSHsm with the COMPACT option. Customers using DFSMSdss with zEDC performing full volume dumps to tape can reduce the number of tape cartridges by up to 19%.

Results measured in an internal lab environment. Your results will vary.
ML1 zEDC Compression – Value!

* Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.

Up to 80% CPU Reduction
Up to 50% Space Reduction
½ the data for SSM
½ the data to Recycle

HSM Migrate to ML1 DASD

- Up to 80% CPU Reduction
- Up to 50% Space Reduction
- ½ the data for SSM
- ½ the data to Recycle

* Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
APAR OA48394 ML2 zEDC Compression

HSM Migrate to ML2 (TS7720 - HYDRA)

- Up to 15% CPU Reduction
- Up to 25% Elapsed Time Reduction
- 40% fewer blocks to Recycle

* Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
ML1 zEDC Compression – Value!

**HSM Recall from ML1 DASD**

- **Up to 65% CPU Reduction**

*Disclaimer:* Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
Content Manager OnDemand can automatically leverage the zEDC Express Accelerator when storing data into the archive as well as when data is requested for retrieval.

Fully compatible with zlib compression used in IBM Content Manager OnDemand today – no changes required at end points.

MIPS reduction of 75% when compared to existing software based compression (results vary by document size and type).

Users currently using compression will see a comparable compression ratio.

---

**OnDemand**

- Long term archive for static content
- Designed for computer generated output (statements, policies, etc.)
- Large data-streams with potentially millions of documents daily
- High-performance batch loading
- Efficient storage and retrieval
- Print data-stream segmentation and indexing

---

Disclaimer: Results based on internal controlled measurements using IBM Content Manager OnDemand and various document types. Results may vary by customer based on individual workload, data, configuration and software levels.
zEDC with z/OS SMF Logger

Alleviate SMF constraints across the entire life cycle of a record using compression technology

Record Creation
- Records created by z/OS, DB2, CICS® and applications are all written to SMF

Record Buffering
- zEDC Express can be used to compress SMF data resident in the z/OS System Logger cutting down on logger storage requirements.

Record Extraction
- SMF can extract compressed data from logger faster than uncompressed data.
  - Targeting a compressed QSAM/BSAM data set for long-term archival can also optimize disk space.
  - zEDC, tailored or generic compression can be used depending on the requirements

Record Archival
- Reading SMF data from a compressed data sets can increase the performance of applications that access that data.
  - Up to 4x less data in System Logger
  - Logger CPU usage reduced by up to 30%
  - Up to 15% reduction in elapsed time for SMF extraction from Logger
  - SMF data stored in zEDC compress BSAM can save up to 4x in archived SMF data size
  - Programs reading SMF data from a zEDC compressed data set can see an elapsed time reduction

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels. (BSAM)
Disclaimer: These results are based on projections and measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels (Logger)
zEDC with WebSphere MQ for z/OS V8

WebSphere MQSeries® has always provided compression options for message data passed over MQ channels via the COMPMSGS 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 COMPMSGS(ZLIBFAST) attribute will now use zEDC when available to perform compression and decompression of message data.

This support is ideal for channels that handle large, 32KB requests.
zEDC with Java

Transparent enablement of the `java.util.zip` package enables high throughput compression and decompression for

- Application Business Data Exchange
- HTTP Responses for Web Services
  - Servlet Filters
  - WebSphere® Web Services component
- Large objects that are serialized and stored
- Exploited through standard Java APIs `java.util.zip`* in the latest releases of Java 7.0.0, and Java V7R1
  - zEDC `java.util.zip.Deflater` in memory test improved elapsed time up to 55x and CPU time up to 240x when compared to zlib software compression.
  - Java application to compress files using `java.util.zip.GZIPOutputStream` class
    - Up to 90% reduction in CPU time using zEDC hardware versus zlib software shown in graphs to the right
    - Up to 74% reduction in Elapsed time using zEDC hardware versus zlib software

* Not all `java.util.zip` classes exploit zEDC

Disclaimer: Results are based on internal controlled measurements using `java.util.zip.Deflater` on data already in memory. Results may vary based on the application’s use of `java.util.zip` classes and other work done by the application
zEDC with Java Application – Encryption Facility for z/OS

Increased throughput and functionality for standard compliant business partner data exchange. The Encryption Facility (EF) for z/OS can now use zEDC to compress and decompress data

1. Before performing an encryption and after performing a decryption
2. As a stand-alone operation

Compressing data with zEDC enables higher throughput than no compression or software compression

zEDC can provide IBM EF users reductions of up to 60% in elapsed time and up to 70% in CPU time for environments where compression is already in use.

For IBM EF users not already using compression, compression with zEDC can provide IBM EF users a reduction of up to 44% in elapsed time and up to 46% in CPU times.

Results based on files containing public domain books. Results may vary by customer based on individual workload, data, configuration, and software levels.

Disclaimer: Results based on internal controlled measurements using IBM Encryption Facility for files containing public domain books. Results may vary by customer based on individual workload, data, configuration and software levels.
Deploying zEDC

- **Operating system requirements**
  - Requires z/OS 2.1 (w/PTFs) and the new zEDC Express for z/OS feature
  - z/OS V1.13 and V1.12 offer software decompression support only

- **Server requirements**
  - Available on zEC12 and zBC12
  - New zEDC Express feature for PCIe I/O drawer (FC#0420)
    - Each feature can be shared across up to 15 LPARs
    - Up to 8 features available on zEC12 or zBC12
  - Recommended high availability configuration per server is four features
    - This provides up to 4 GB/s of compression/decompression
    - Provides high availability during concurrent update (half devices unavailable during update)
    - Recommended minimum configuration per server is two features
  - Steps for installing zEDC Express in an existing zEC12/zBC12
    - Apply z/OS Service; Hot plug a zEDC Express adapter; IODF updates and Dynamic Activate

- **Capacity Planning**
  - The IBM System z Batch Network Analyzer tool reports on potential zEDC usage for QSAM/BSAM data sets

* For the full zEDC benefit, zEDC should be active on ALL systems that might access or share compressed format data sets. This eliminates instances where software inflation would be used when zEDC is not available.
zEDC and Resource Groups

- During Resource Group level microcode updates half of the zEDC adapters are brought offline during the update.
- To avoid a temporary single point of failure, four zEDC devices can be deployed across two resource groups.
IBM System z Batch Network Analyzer and Compression

Helping determine if you have files that are candidates for zEDC

- IBM System z Batch Network Analyzer
  - A free, “as is” tool to analyze batch windows
  - Available to Customers, Business Partners and IBMers
  - PC based, and provides graphical and text reports
    - Including Gantt charts and support for Alternate Processors

- Available Now on TechDocs
  https://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS5132

- zBNA identifies zEDC Compression Candidates
  - Post-process customer provided SMF records, to identify jobs and data sets which are zEDC compression candidates across a specified time window, typically a batch window
  - Help estimate utilization of a zEDC feature and help estimate the number of features needed
  - Generate a list of data sets by job which already do hardware compression and may be candidates for zEDC
  - Generate a list of data sets by job which may be zEDC candidates but are not in extended format
zEDC RMF Reporting

New RMF™ report shows the utilization of each device.

**RMF Postprocessor Interval Report: PCIE Activity Report**

<table>
<thead>
<tr>
<th>Function ID</th>
<th>Time Busy %</th>
<th>Request Execution Time</th>
<th>Std Dev for Request Execution Time</th>
<th>Request Queue Time</th>
<th>Std Dev for Request Queue Time</th>
<th>Request Size</th>
<th>Transfer Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0013</td>
<td>0.889</td>
<td>7.78</td>
<td>0.417</td>
<td>15.0</td>
<td>0.953</td>
<td>24.3</td>
<td>21.5</td>
</tr>
</tbody>
</table>

**Hardware Accelerator Activity**

The percent of this interval where this specific zEDC Express device was executing requests.

Compression ratio of all requests serviced by zEDC. This will span all usage of the zEDC Express devices.

Average request queue time in Microseconds for this device.
zEnterprise Data Compression Redbook

- Provides overview of the technology
- Covers configuration of SMF, QSAM/BSAM and DFHSM/DFDSS
- Examples using zBNA
Thank You!
What is DEFLATE

The DEFLATE file format is defined by the IETF RFC1951 document. The generation of the DEFLATE data is up to each implementer.

There is a combination of two processes:

LZ77 (Lempel-Ziv 1977) – Provide pattern matching via a 32k rolling window in the data. As matches are found they are replaced with a back reference to the match.

Huffman Coding – Encodes the symbols in the file into a set of bit patterns where the most used symbols get the smallest bit patterns.

There are two types of approaches for Huffman Coding

Static or Fixed Huffman – A predefined alphabet is used to encode the symbols. This alphabet is defined in RFC1951.

Dynamic Huffman – The Huffman Tree is defined based on the symbols in the stream. The Huffman Tree alphabet is embedded in the DEFLATE block.

The file format is a BIT aligned file; meaning that symbols do not fall on byte boundaries.
What is a *Rolling Window*

The DEFLATE compression format is considered **dictionary-less**. That means that the compressor of the file does not have to also save a dictionary of potential string matches. This is accomplished with DEFLATE because the uncompressed data is the dictionary.

Take for instance a distance/length pair in the input compressed data stream.

After this symbol is decompressed, the rolling window looks like the following for the next symbol evaluation:
### DEFLATE Example

#### Compressed Data

<table>
<thead>
<tr>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar bar bar</td>
</tr>
<tr>
<td>boo boo boo</td>
</tr>
<tr>
<td>blah blah blah</td>
</tr>
<tr>
<td>foo boo blah</td>
</tr>
<tr>
<td>blah foo blah</td>
</tr>
<tr>
<td>hello</td>
</tr>
</tbody>
</table>

The 1st backwards reference length includes symbols that are part of the compressed symbols themselves as shown here:

#### Byte #  Codes and Values

<table>
<thead>
<tr>
<th>Byte #</th>
<th>Codes</th>
<th>Value</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00b2 = 0082 ('b')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000001</td>
<td>00b1 = 0081 ('a')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000002</td>
<td>0199 = 0099 ('r')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000003</td>
<td>0070 = 0040 (' ')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000004</td>
<td>00b2 = 0082 ('b')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000005</td>
<td>&lt;000a, 0004&gt; = 0081 0099 0040 0082 0081 0099 0040 0082 0081 0099</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000015</td>
<td>0045 = 0015 ('\n')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000016</td>
<td>00b2 = 0082 ('b')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000017</td>
<td>0196 = 0096 ('o')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000018</td>
<td>0196 = 0096 ('o')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000019</td>
<td>0070 = 0040 (' ')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000020</td>
<td>&lt;000b, 0004&gt; = 0082 0096 0096 0040 0082 0096 0096 0040 0082 0096 0096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000031</td>
<td>0045 = 0015 ('\n')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Backward Reference

<table>
<thead>
<tr>
<th>Byte#</th>
<th>String</th>
<th>Len</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>bar-b</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>5</td>
<td>bar-ba</td>
<td>1</td>
<td>'a' at byte 1</td>
</tr>
<tr>
<td>6</td>
<td>bar-bar</td>
<td>2</td>
<td>'r' at byte 2</td>
</tr>
<tr>
<td>7</td>
<td>bar-bar-</td>
<td>3</td>
<td>' ' at byte 3</td>
</tr>
<tr>
<td>8</td>
<td>bar-bar-b</td>
<td>4</td>
<td>'b' at byte 4</td>
</tr>
<tr>
<td>9</td>
<td>bar-bar-ba</td>
<td>5</td>
<td>'a' at byte 5</td>
</tr>
<tr>
<td>10</td>
<td>bar-bar-bar</td>
<td>6</td>
<td>'r' at byte 6</td>
</tr>
<tr>
<td>11</td>
<td>bar-bar-bar-</td>
<td>7</td>
<td>' ' at byte 7</td>
</tr>
<tr>
<td>12</td>
<td>bar-bar-bar-b</td>
<td>8</td>
<td>'b' at byte 8</td>
</tr>
<tr>
<td>13</td>
<td>bar-bar-bar-ba</td>
<td>9</td>
<td>'a' at byte 10</td>
</tr>
<tr>
<td>14</td>
<td>bar-bar-bar-bar</td>
<td>10</td>
<td>'r' at byte 11</td>
</tr>
</tbody>
</table>

#### DEFLATE Example

<table>
<thead>
<tr>
<th>Start of overlap</th>
</tr>
</thead>
</table>

Backward reference: length=10, distance=4 ('a' character at byte# 5)
The zlib Library

zlib is a widely used open source C library that provides compression and decompression. It supports RFC1950 (ZLIB), RFC1951 (DEFLATE), and RFC1952 (GZIP).

zlib supports a streaming model such that files can be compressed/decompressed in chunks.

Things to know about using zlib with zEDC

- The address space using zEDC needs READ access to the FPZ.ACCELERATORS.COMPRESSION SAF resource
- The FIRST deflate() or inflate() request must be at least as large as the minimum threshold setup in IQPPRMxx
- The window size for deflate must be 32k
- The _HZC_COMPRESSION_METHOD environmental variable can be used to force software even when zEDC is available
The IQPPRMxx member in SYS1.PARMLIB can be used to adjust internal settings for zlib behavior. For example

```
ZEDC,DEFMINREQSIZE=4,MAXSEGMENTS=7
```

This member will set the minimum input size of a deflate request via zlib to 5Kb and will set the maximum internal buffer size to 7 16Mb segments. The current settings and buffer usage can be displayed with the D IQP command:

```
D IQP
IQP066I 10.12.37 DISPLAY IQP 364
zEDC Information
    MAXSEGMENTS: 7  (112M)
    Previous MAXSEGMENTS: 4  (64M)
    Allocated segments: 1  (16M)
    Used segments: 0   (0M)
    DEFMINREQSIZE: 4K
    INFMINREQSIZE: 16K
    Feature Enablement: Enabled
```

Updated values from IQPPRMxx
The zlib Internal Structure

If the request can not be performed using zEDC then the software zlib code will be used for the request.

The zEDC requests are not done directly from user storage; the input data is copied into pre-allocated buffers and the output data is copied from these buffers back to user storage.

- C Program
- Java Program using Java.util.zip package
- z/OS zlib library
- Standard S/W Zlib
  - If zEDC devices and request eligible
  - Else use S/W algorithm
- Pre-pinned z/OS Memory
- 4K to 512K
- Caller Input Buffer
- Caller Output Buffer
- zlib State Buffer
- zEDC Device
zlib Input Buffer Size Matters

![Graph showing the relationship between Request Size (Bytes) and Time for different types of elapsed and CPU times.]

Disclaimer: Based on projections and/or measurements completed in a controlled environment. Results may vary by customer based on individual workload, configuration and software levels.
Java 7.1 provides zEDC access via the `java.util.zip Inflater` and `Deflater` classes. The same conditions that apply to zlib also apply to Java.

This example (try/catch blocks removed) shows the critical buffer sizes. APIs that are accelerated:

1. `java/util/zip/Inflater`
2. `java/util/zip/Deflater`
3. `java/util/zip/DeflaterOutputStream`
4. `java/util/zip/GZIPInputStream`
5. `java/util/zip/GZIPOutputStream`

Data is read from an uncompressed file and written to a compressed file.

```java
byte buffer[] = new byte[64 * 1024];
byte outputFile[];

input = new FileInputStream(argv[0]);
output = new ByteArrayOutputStream();
gzStream = new GZIPOutputStream(output, 4096);
for(;;) {
    readBytes = input.read(buffer);
    if (readBytes < 0) {
        break;
    } else {
        gzStream.write(buffer, 0, readBytes);
    }
}
```

64Kb input buffer for `deflate()`. This must reach the threshold.

4Kb output buffer for `deflate()`.
Setting Buffer Sizes using Java

For the Inflater and Deflater classes the *input buffer* size is the size of the parameter passed via the `setInput` method.

For the GZIPInputStream, DeflaterInputStream and InflaterInputStream classes a constructor is provided which allows the *input buffer* size for the deflate or inflate operation to be specified. The buffer passed to the `read` method determines the size of the *output buffer*.

For the GZIPOutputStream and DeflaterOutputStream classes a constructor is provided which allows the *output buffer* size for deflate and inflate operations to be specified. For these classes the size of the buffer passed to the `write` method sets the *input buffer* size.
zEDC SMF30 Updates

New zEDC section in the SMF30 record provides per-job usage statistics

– Shipped with OA45767, also track OA48268 (currently open)

– The following statistics are provided in the new section

  • Number of requests
  • Total Queue Time
  • Total Execute Time
  • Uncompressed Input Bytes
  • Compressed Output Bytes
  • Compressed Input Bytes
  • Uncompressed Output Bytes
Ordering Considerations for the z/OS V2.1 zEDC Express for z/OS feature

To use zEDC, you need to order both the zEC12 or zBC12 H/W feature and the z/OS V2.1 priced feature. This is a new delivery model, that is designed to help with the overall TCA and designed to scale as you expand your use of zEDC.

z/OS V2.1 considerations for the new zEDC Express for z/OS feature:

- eConfig will notify you that there is a required S/W feature at the time you configure the zEDC Express feature for PCIe I/O drawer (FC#0420)
- Similar to other z/OS priced features, it:
  - is MLC-based (Monthly License Charged)
  - should be licensed only to those servers that will be exploiting zEDC
  - should be licensed to your server(s) at the time you are ready to start using zEDC. This will ensure that billing will align with the planned exploitation of zEDC.
    - Contact TechLine or your MLC pricing representative for details.
DSS/HSM Support for zEDC compressed format data sets

- Supported in all **DFSMStss** functions
  - COPY, DUMP, RESTORE, CONSOLIDATE, DEFRAG, PRINT, RELEASE
  
  Note: When copying or restoring a data set the compression type of the source is carried forward to the target. Data remains in compressed form on COPY, DUMP, RESTORE

- Supported in all **DFSMShsm** functions
  - MIGRATION, RECALL, BACKUP, RECOVER, FULL-VOLUME DUMP, RECOVER FROMDUMP, FRBACKUP, FRRECOV, ABACKUP, ARECOVER
Converting existing QSAM/BSAM to zEDC Compressed

To take advantage of zEDC compression, existing 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 re-compressing target
- Standard utilities can be used to perform the copy, for example
  - ISPF 3.3 Copy data set
  - IDCAMS REPRO
  - IEBGENER

IEBGENER Example where SYSUT1 can be any sequential data set (compressed or non-compressed) and SYSUT2 is created with a DataClass containing COMPACTION=ZR (zEDC Required) or ZP (zEDC Preferred)

```
//SMITH2   JOB 1,GEOFF,MSGCLASS=X
//         EXEC PGM=IEBGENER
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*  
//SYSUT1   DD DISP=SHR,DSN=SMITH.SEQ.DATA  
//SYSUT2   DD DISP=(NEW,CATLG),  
//         DSN=SMITH.ZEDC.DATA,STORCLAS=SCZEDC  
//         DATACLAS=DCZEDC,SPACE=(CYL,100,10))
```

NOTE: DFSMSdss COPY does not convert to zEDC compressed format since the target data set retains characteristics of source data set.