PKWARE®

DATA CENTER PERFORMANCE GUIDE:

Four Ways to Do More with Less

JOE STURONAS

CHIEF TECHNOLOGY OFFICER, PKWARE

Contents

Letter from the Author

As stewards of the information technology assets of our organization we know there is constant pressure to improve performance, find more efficient storage techniques and reduce costs. As CIOs, data center administrators, infrastructure architects and application developers this is often easier said than done.

The good news is that there is a steady stream of new performance technologies that can help. Over the past few years IBM has released some new capabilities that exploit several mainframe hardware facilities to offload compression and free up general computing capacity with the goal of reducing the total cost of computing. These are exciting breakthroughs that provide 10x speed improvements that were not previously possible. And we believe this changes the discussion about how IBM's mainframes can provide blazing speeds where fast data throughput is required while also squeezing more cost out.

We highlight these new performance technologies and share input from customer experiences in this data center performance guide. The best practices and real-world use cases in the guide are meant to spark some ideas that can help you get the most out of your data center. No two data centers are the same. However, we're hopeful this guide will also act as a conversation starter on your unique performance challenges.

JOE STURONAS CHIEF TECHNOLOGY OFFICER, PKWARE
joe.sturonas@pkware.com

DATA CENTER PERFORMANCE GUIDE:

Four Ways to Do More with Less

We recently surveyed our customers on ways to get the best performance out of their data center environments. We got an earful! It wasn't a surprise that the most common theme in the data center was optimization and reducing costs. At the same time, data center operators face substantial challenges like the surge in unstructured data. Unstructured data counts for approximately 80% data center storage. On top of that, constrictive Service Level Agreements (SLAs) must be met while there are restrictions on further investments in infrastructure hardware and software.

Working with our customers, we've targeted four areas to help optimize data center environments without taking on huge, new expenditures or programs. The four areas are:

1. Efficient use of bandwidth
2. Improved data storage
3. Ease of file management
4. Efficient use of transfer between platforms

In this performance guide we will explore in detail the many ways that we've worked with customers to get the most from their IBM® System z® and data center environments.

1 Efficient Use of Bandwidth

Often the perception is that we live in a world with unlimited bandwidth. The reality in the data center is that, in fact, we are bandwidth constrained. A chain is only as strong as its weakest link and in networking, bandwidth is measured by the speed of the slowest hop. SLAs are measured against the end-to-end time it takes to deliver data to customers, partners and constituents.

One primary strategy to free up bandwidth is to use data compression. Compressing data before it is sent to the consuming application can greatly improve the efficiency of transferring data. However, because CPUs and storage have become so fast, the time to compress data with CPU cycles far exceeds the time to transfer the data. As a result, many mainframe shops have been reluctant to consider compressing data before transferring files due to concerns over how compression counts against their general CPs. However, new software technologies and architectures can now take advantage of special purpose engines and specialty PCIe cards making compression a legitimate, nearly free resource, to open up bandwidth.

WHY IS COMPRESSION CPU INTENSIVE?

To explain why compression can be very CPU intensive, consider an example of a simple data compression technique known as Run-Length Encoding method. This method works when repeating characters are evident in a data stream. The run of characters is represented in a compressed form as a single character with its count.

Example: `B 2 2 2 2 E H H H H H H H H H`

Compressed: `B 2*4 E H*9`

However, to perform a thorough compression operation, more advanced algorithms and enhanced techniques are required which work at the bit level and allow for noncontiguous iterations of bit strings, such as the Deflate compression algorithm, created by the PKWARE founder Phil Katz.

The compression algorithm Deflate is able to achieve significant compression ratios, in many cases around 90%, because it uses a sliding window dictionary. A sliding window dictionary is constantly looking at the input data, and works to improve the dictionary to find the optimal replacement strings that will yield the best compression ratios. This is why the Deflate compression algorithm is so CPU intensive, but also why it produces the best compression ratios, even when the profile of the input data can change significantly from the beginning of the file to the end of the file.

The opposite of Deflate is Inflate. Inflate is the reverse process of replacing the tokens with the longer string patterns and restoring the file to its original content and size. The Inflate algorithm is very efficient and not very CPU intensive because it is simply doing a lookup and replace.

COMPRESSION IN HARDWARE

The ability to perform hardware compression on System z has existed since the mid-90's using the IBM Data Compression Services through the assembler instruction CMPSC. The CMPSC compression facility is very efficient and uses hardware that is built into all modern System z mainframes. The CMPSC does not use the Deflate compression algorithm, but instead uses a static dictionary. Using a static dictionary for compression is not CPU intensive at all, because it is not working to improve the dictionary during the compression process

based on the input data. The downside of using static dictionary compression is that the compression ratios are on average going to be significantly less than using a sliding window dictionary algorithm like Deflate, unless the static dictionary is built specifically for the file it is compressing.

CMPSC static dictionary compression can be effective if a static dictionary can be built for an application profile of datasets, such as log files that contain predictable patterns of data. The specific static dictionary would only be effective on that application profile of datasets, but still most likely only capable of approximately a 70% compression ratio, whereas Deflate might be capable of a 90% compression ratio.

In 2006, IBM introduced the System z Integrated Information Processor (zIIP) which was initially intended to offload DB2 workloads, but has been expanded to include other z/OS workloads allowed by IBM from other ISVs. The zIIP is available on all IBM zEnterprise™, System z10, and System z9 servers. It is designed to help free-up general computing capacity and lower overall total cost of computing for select data and transaction processing workloads (FIGURE 1). IBM will not impose software charges on zIIP capacity. Therefore, Using a zIIP engine to perform Deflate compression can have dramatic performance improvements in most circumstances.

You might be surprised that PKWARE's PKZIP and SecureZIP for z/OS (version 14 and higher) now support

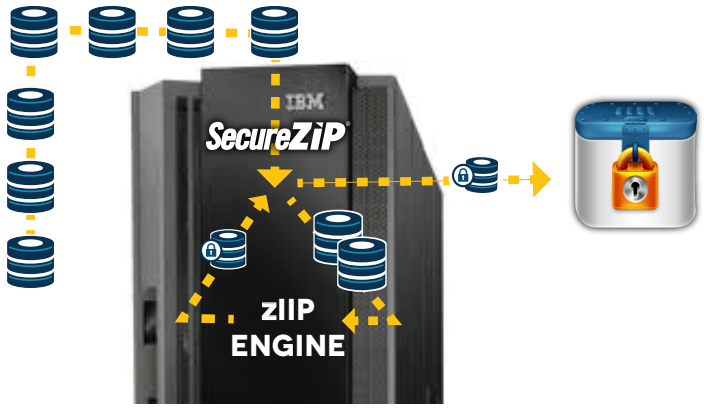


FIGURE 1

1. Data Streams through the SecureZIP process.
2. Data is offloaded, encrypted and compressed
3. Results in compressed and encrypted data without increasing mainframe capacity.

the zIIP processor. PKZIP and SecureZIP for z/OS have made compression and CRC (Cyclical Redundancy Check) zIIP eligible. This means that when PKZIP and SecureZIP for z/OS are compressing a file, approximately 90% of the CPU workload is zIIP eligible and only 10% is processed by the general CPs. Anytime that special purpose engine can be used over the general purpose CPs, it is like free processing. Once the initial cost of the special purpose engine has been paid for, more processing can be put on that engine, more CPs can be saved and more costs that can be recovered.

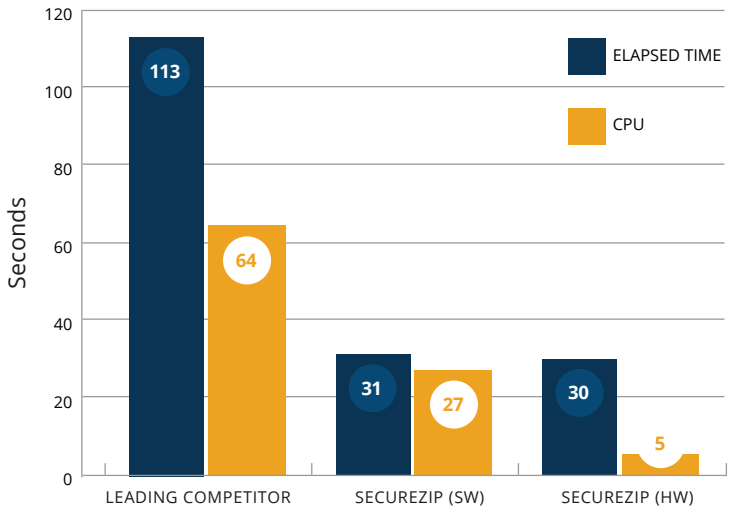
The chart below (FIGURE 2) shows the relative comparison benchmark between different applications that utilize the zIIP hardware.

FIGURE 2

Compression Benchmark Comparision

2GB BINARY FILE

RUNNING ON ZEV12 WITH ZIIP

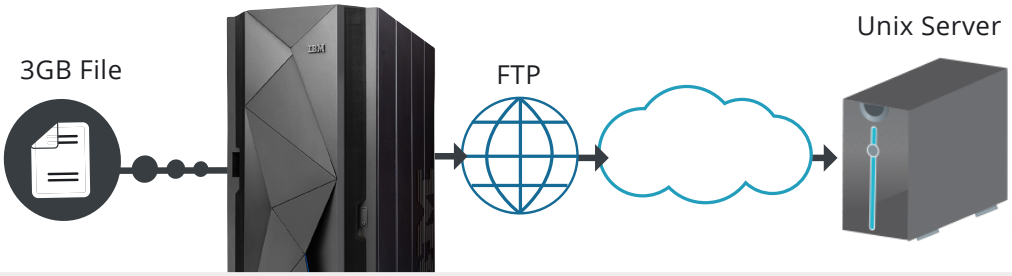


The IBM zIIP special purpose engine can also have a dramatic impact on storage and CPU time. The above chart reflects a customer benchmark of compressing a 2GB file with PKZIP for z/OS with and without hardware assistance for the compression operations. Without any assistance from a zIIP engine, our customer observed an elapsed time of 31 seconds and a CPU time of 27 seconds (all using the General CPs).

Compressing that same 2GB file using a zIIP special purpose engine by making the Deflate compression algorithm and CRC (Cyclical Redundancy Check) zIIP eligible, PKZIP for z/OS v15 were able to reduce the total general CP down to 5 seconds. That resulted in an 81% reduction in chargeable CPU over the PKZIP and for z/OS software based solution and a 92% improvement in CPU time over the leading competitor.

ZIIP HARDWARE COMPRESSION IN FILE TRANSFER

FIGURE 3
Overall 5 hours elapsed time over a T1 (1.5 Mbps)
without compression

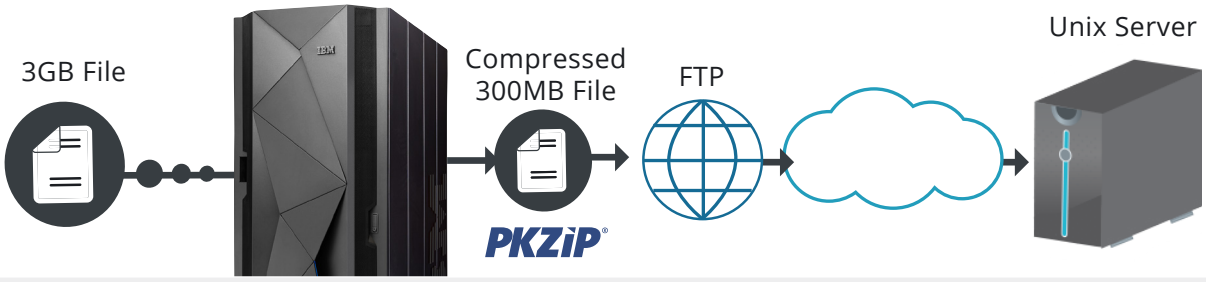


Using a zIIP engine to perform compression can have dramatic performance improvements in many circumstances. Another example is a customer running on a z10-BC with 4 general CPs that were all knee capped at about 25% of their total capacity each, totaling about 731 MIPS and approximately the capacity of an un-kneecapped uni. They were trying to move a 3GB file on a nightly basis. They had to compress it because the bandwidth to get the file to their partner was constrained at 1.5Mbps, which would take almost 5 hours if it was uncompressed (FIGURE 3). They needed to get the file moved in 3 hours.

The problem was that their general CPs were maxed out at 100% during the batch window when this file needed to be transferred. They did not have the capacity with the general CPs to compress the data to 90% of its original size so that the file could arrive at the partner in 30 minutes. This customer did have a zIIP special purpose engine because it was used for DB2 DRDA during the day, but the zIIP was hardly utilized during their batch window.

By using a revised compression strategy, they were able to offload 90% of the CPU to the zIIP engine without impacting the already fully utilized general CPs. Thus, they were able to compress the file in approximately 1 elapsed minute and reduce the file size from 3GB to 300MB (FIGURE 4). It then only took less than 30 minutes to transfer the 300MB file to the partner using the existing file transfer step they had in place. They were able to reduce a process that was taking well over 5 hours down to 30 minutes, and comfortably met their 3 hour service level agreement with the partner. They did all

FIGURE 4
Overall
30 minutes
elapsed time
over 1.5 Mbps
with compression



this without having to increase the chargeable capacity of their z10-BC, which would have meant costs for the hardware upgrade as well as software charges for the increased capacity.

Adding compression to a file transfer workflow does not require sophisticated modifications or program changes. The level of abstraction is at the file level and the file or files can be compressed at the penultimate step before transmission.

Managed file transfer programs on z/OS provide optional file compression, but typically do so very inefficiently. z/OS managed file transfer programs provide a very sophisticated and full featured facility, but they lack the ability to compress using the zIIP specialty engine and only utilize the general purpose CPs to perform Deflate compression on the files.

ZEDC HARDWARE ASSISTED COMPRESSION

Earlier this year, IBM introduced a new PCIe card called zEnterprise Data Compression (zEDC). This sole purpose of this card is to perform deflate compression. It does compression very quickly and scales nicely. The zEDC requires some of the latest hardware and software because it requires a zBC12 or a zEC12 with GA2 microcode. The zEDC also requires z/OS v2.1, which is the latest version of z/OS that went GA in September 2013.

Like the zIIP special purpose engine, the zEDC is not considered as chargeable capacity like a general CP, but it can provide a very specialized workload capability for compression. The zEDC is very similar to the Crypto Express cards (CEX3C, CEX4C), which are also PCIe cards that contain their own processor and perform specialized cryptographic work that is off board processing from the CPs. The main function of the zEDC is to compress a buffer. Applications like PKZIP and SecureZIP for z/OS v15 support compression with the zEDC. The zEDC supports the Deflate compression algorithm, so the compression that is done with the zEDC can be inflated on other platforms that support Deflate. The zEDC is very scalable, and provides a tremendous amount of throughput since many LPAR's can access the same zEDC and most likely not saturate it.

FIGURE 5: zEDC

FIGURE 6

ZEDC HARDWARE COMPRESSION IN FILE TRANSFER

A financial services company with a more significant amount of data to send to their partners was also running on a z10-BC W05 (approximately 2000 MIPS in total) with a zIIP specialty engine. They were missing their SLAs for getting large reports to partners where they had a 3 hour delivery window and it was taking 3 hours to move 100 files totaling 5GB (FIGURE 6). This allowed no room for error, and invariably, jobs fail and need to be restarted. If there was just one error, they missed their SLA. They were fined $1,000 for each missed SLA and those fines were accumulating to about $10,000/month. Their z10-BC was at peak capacity during the time they needed to perform the file transfer to their partners using Connect:Direct.

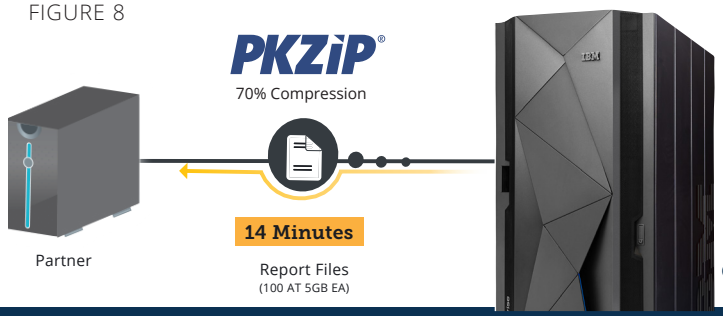
After compression was inserted into the workflow, they were able to take advantage of their under utilized zIIP special purpose engine and remove the burden of the already taxed general CPs and compress the files to a 95% compression ratio, reducing the total size to 250MB (FIGURE 7). The transfer now took only 15 elapsed minutes to compress the files and 10 minutes to transfer the files. PKZIP was added as additional steps to the existing file transfer job, without any changes to the applications creating the data itself. PKZIP integrates at the file level abstraction layer. The process that was once taking 3 hours now is reduced to 25 minutes. This allows for a comfortable cushion for any restarts or delays they might experience without missing their SLA's. Because 90% of PKZIP's overall CPU time was offloaded to their zIIP engine, they had no need for additional capacity.

This customer was also on the IBM Early Ship Program for the zBC12 running z/OS v2.1. Along with the zIIP special purpose engine, they installed a zEDC. PKZIP for z/OS v15 has the concept of a compression facility, meaning it uses the most efficient compression facility that is available in the environment they are running on. It could use the zEDC, the zIIP or fall back to the default software. Because of this, the customer did not have to modify any of their existing JCL because PKZIP auto-detected the zEDC and was able to take advantage of it.

In this use case, the zEDC was able to compress the files down to a 70% compression ratio (FIGURE 8). That is less than what was possible with the zIIP engine or

FIGURE 7

FIGURE 8



software-only because PKZIP has 9 levels of compression and the zEDC only supports 1 level of compression. The trade off is there is much more throughput on the zEDC, so the overall elapsed time is reduced to about 1 elapsed minute. The additional size means that it takes 13 minutes to transfer the data, as opposed to 10 minutes. Overall, the 3 hours that were reduced to 25 minutes are now further reduced to about 14 minutes. This allows for many restarts and delays without missing any SLAs. Like the zIIP, this approach does not place any computational burden on the general CPs.

The following is a discussion about addition strategies for creating efficient use of bandwidth.

Data file transmissions typically extend across multiple operating environments such as mainframe-to-UNIX/Linux/Windows® Server/Desktop, mainframe-to-mainframe, and mainframe-to-IBM i. PKZIP and SecureZIP for z/OS are built to support the EBCDIC-to-EBCDIC and EBCDIC-to-ASCII data translation that is crucial to the formatting and processing of data exchanged across platforms. By default PKZIP and SecureZIP will auto-detect the input data format and perform any necessary data translation. However, the following best practices should be considered when deploying PKZIP or SecureZIP as part of your data transmissions.

Mainframe-to-Mainframe/IBM i (EBCDIC-to-EBCDIC)

Eliminate data translation during the compression routine.

- Specify `BINARY` or `BINBLK` as the `DATA_TYPE`
- Use `SAVE_LRECL(Y)`

Using these parms/settings will allow PKZIP and SecureZIP to retain the native EBCDIC format of the data and DCB attributes. This will also eliminate any CPU overhead related to auto-detecting if the input datasets need to be translated to ASCII during data compression.

Mainframe-to-Server/Desktop (EBCDIC-to-ASCII)

Eliminate data translation during the compression routine.

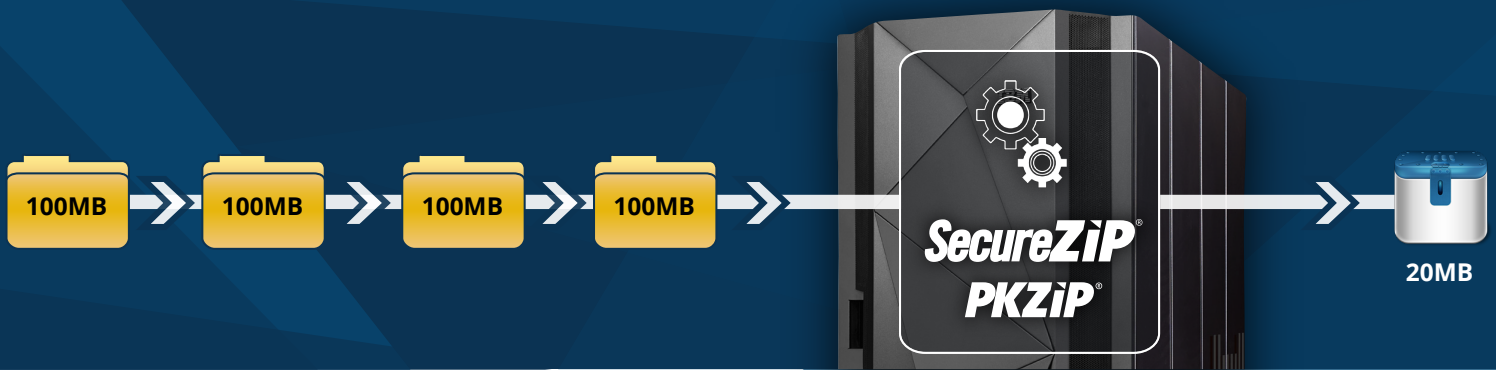
- Specify `TEXT` as the `DATA_TYPE`
- Modify `DATA_DELIMITER` to consider the UNIX/Linux `(0A)` or Windows `(0D0A)` carriage return life feed options
- Add `FILE_TERMINATOR()` so there's no file terminator at the end of the data stream

By specifying the TEXT option, PKZIP and SecureZIP do not have wasted cycles to decipher whether the data should be translated to ASCII, or stay as EBCDIC, eliminating unwanted CPU overhead.

PKZIP and SecureZIP for z/OS also provide users the ability to rename input and output datasets in anticipation of the respective destination platform. When transmitting between mainframe environments, the sending/receiving environments will likely have different high-level qualifiers and node structures. So users can utilize the following parameters to rename input/output datasets to an environment friendly HLQ and/or DSN …

- `ZIPPED_DSN (PKZIP)`
- `UNZIPPED_DSN (PKUNZIP)`

These parms have been essential for clients exchanging data between a mainframe and UNIX/Linux/Windows infrastructure that support long filenames. With dataset names capable of 8 character nodes, decompressing a file received from a Windows environment typically will not have the DSN node structure, but instead have a longer filename with an extension (i.e. Client_A_Report.csv). The UNZIPPED_DSN parm allows users to rename the received file during decompression to an IBM friendly DSN/node structure (i.e. APP.CLIENTA.REPORT.CSV). The same is possible in the opposite direction, allowing users to rename the input DSN to a UNIX/Linux/Windows friendly filename during the compression routine.



2 Improved Data Storage

Compression using the zEDC differs from compression using the zIIP in that workload targeted for the zEDC is guaranteed to execute on the zEDC. The zIIP is designed so that a program can work with z/OS to have a portion of its enclave service request block (SRB) work directed to it. For example, compression workloads are those executing in enclave SRBs which are eligible for zIIP. The z/OS Workload Manager (WLM) determines whether the enclave SRB work should be processed by the zIIP.

As the zEDC only requires a negligible amount of general processing, there isn't the same capacity burden that compression once placed on a mainframe environment's general CP. Files that need to be archived can be compressed so that they have a very small footprint either on mainframe storage or off mainframe storage such as a NAS.

Storing all the metadata about a file being compressed in a ZIP archive itself, for instance, means extracting that same file at a later time does not require the file first be allocated before it is extracted. PKZIP can take the metadata, such as the DCB (LRECL, BLKSIZE, DSORG), SPACE or SMS information, and perform dynamic allocation of the dataset and then extract the data in the same step automatically.

There is an interesting performance benefit of storing metadata in an archive format like ZIP for z/OS data. Any type of mainframe data can be archived anywhere, including public cloud storage, such as Amazon S3.

As another example, due to compliance and regulatory purposes, a mainframe customer was required to store their proprietary transaction logs for several years. These logs were stored in a VSAM ESDS and each day they produced approximately 80GB (approximately 2TB a month). It seemed rare that they would ever need to access the data, but for regulatory purposes they needed to retain the data.

They had been storing the data on tape which required about one 3592 tape each week. Using an archive format like ZIP, provides the alternative of retaining these archives to disk rather than to tape. The customer did not have to keep any hardware around for the time the old media would expire from a compliance perspective. Retaining the information on disk allows them to be flexible in terms of where the data is retained and allows it to be moved easily. And, storing it in the ZIP format with PKZIP allows the data to be extracted on any of the other major platforms for later use while meeting the compliance requirement.

FIGURE 9

Mainframe datasets like VSAM ESDS's can be stored in an archive, where all the metadata about the VSAM dataset is stored in the archive, and then transported to any platform, including cloud storage such as Amazon S3. Because the data was being stored in public cloud storage, the customer needed to encrypt the data using AES 256-bit encryption. This provides protection for the data even in the public cloud.

If the VSAM ESDS ever needs to be extracted with SecureZIP, there is no need to pre-allocate the VSAM ESDS with IDCAMS. Given that the metadata about the VSAM dataset is in the archive, SecureZIP is able to perform dynamic allocation for the VSAM dataset just prior to extraction.

APPLICATION INTEGRATION

Most of the time abstracting at the file level is the simplest and most straight forward way to include compression and encryption. This is because this approach does not require any program changes and does not alter the existing workflow of information; it simply augments it.

In the case where performance is critical, file abstraction might not be the best choice. Instead, modifying programs to interface with compression and encryption facilities at a binary abstraction rather than a file abstraction interface will be more effective. Virtual Objects is a way to achieve significantly higher level of performance.

The Virtual Objects Interface extends the Basic Call Mode interface by allowing the use of memory objects in the place of ZIP archive files and application data or files. You can combine file-based processing Basic Call Mode with Virtual Objects mode to gain flexibility and performance, but processing ZIP data directly from and to memory objects allows the allocation to process without the overhead of file based archives and application data. Various combinations of file objects and memory objects make this interface much more flexible and can easily satisfy many different application requirements. Multiple objects can be processed in a single call to ZIP which can produce a significant reduction in elapsed time. Additionally, extended status information is also available to the application through the interface providing runtime statistics and error information.

This diagram illustrates how using the file level abstraction Interface, the application data that resides in DB2 must be compressed, encrypted and sent to another application or end user, in this case via MQ Series. The data retrieved by DB2 must be written to a data set. SecureZIP is then called to compress and encrypt that data set into an archive. Then the archive is read by the application program and placed in an MQ data buffer which is then sent to its destination by MQ Series services. This introduces 4 sets of file I/O's for each of

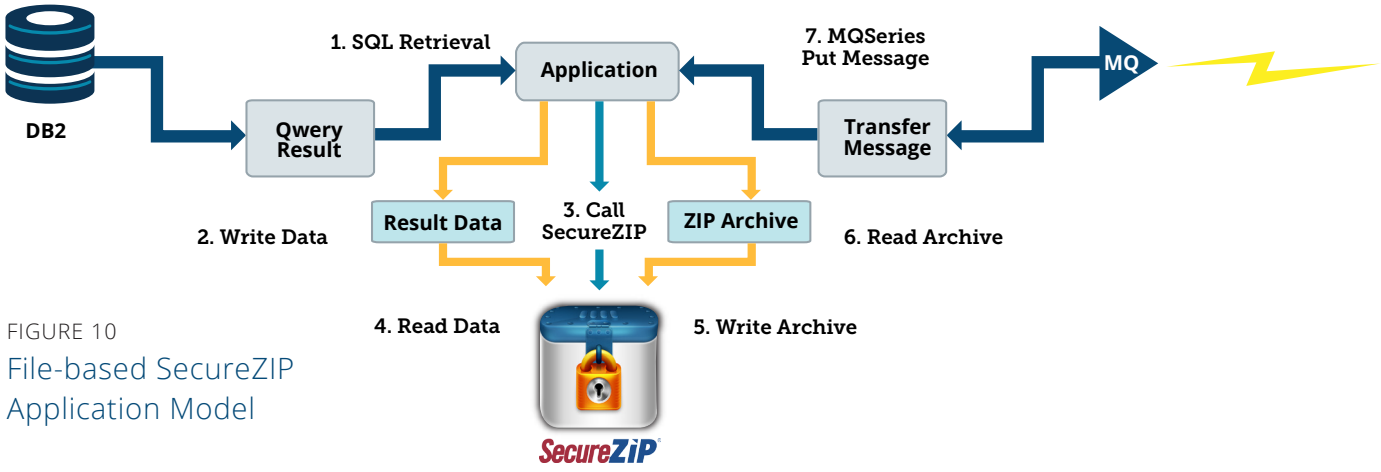


FIGURE 10
File-based SecureZIP Application Model

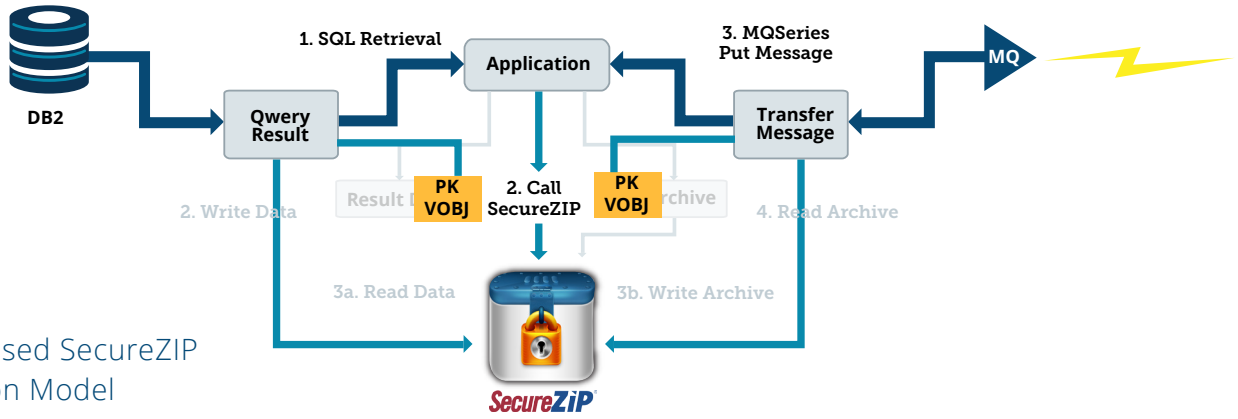


FIGURE 11
Object-based SecureZIP Application Model

the archive sessions. This is a very simple model which works, but could be improved by reducing the number of I/O's.

Using the Virtual Objects Interface, the memory area containing the DB2 SQL result is passed directly to SecureZIP by placing the memory address of the data in a virtual object. SecureZIP is called via a standard program call. It processes the data and places the resulting archive in the memory location addressed in a second object that was passed on the call. Archive size information is passed back to the calling program via a third object; the session object. This implementation allows SecureZIP to utilize existing application memory areas and eliminate 4 of the 7 steps above in the file abstraction example.

ARCHIVING BEST PRACTICES

When archiving datasets, the most efficient compression performance is gained by again eliminating any data translation during the compression routine. To do so, the user should specify BINARY as the DATA_TYPE, as well as the SAVE_LRECL parameter set to YES (Y). Leveraging these parms/settings will allow PKZIP and SecureZIP to retain the native EBCDIC format of the data and DCB attributes, and eliminate any CPU overhead related to auto-detecting if the input datasets need to be translated. By retaining the EBCDIC structure and DCB attributes during data compression, there is no need to pre-allocate the datasets prior to them being decompressed. PKZIP and SecureZIP automatically detect the attributes, using them to restore the output dataset(s) to their original state (BLKSIZE, LRECL, RECFM, etc.). As noted above, the UNZIPPED_DSN can be leveraged to modify the output DSN as necessary upon decompression of the datasets stored inside of the ZIP container.

3 Ease of File Management

A global logistics company was transferring thousands of file manifests for their international shipments to the tune of about 5,000 individual file transfers a day. In those transfers, there was a mean time between failure of approximately 2,500 file transfers or about two failures a day. The more file transfers, the more room for error and the more transmission failures. When a failure occurred a Production Support Analyst would need to investigate the error and restart the transfer with the appropriate restart parameters so the file transfers could continue processing.

This customer installed PKZIP for z/OS and batched up the files they needed to transfer into 25 files per 1 ZIP file. Because they were able to compress the files, the size of the files to transfer was much smaller. More significantly, the possibility of transfer failures was cut from daily occurrences to a few times a month.

FILE MANAGEMENT BEST PRACTICES

Whether your data files are stored on the current mainframe environment, or transferred to others, it's important to understand the size and type of the data being added to the ZIP archive. PKZIP/SecureZIP can compress tens of thousands of files into a single ZIP container as detailed above, but there are also performance considerations to make when doing so.

- Specify `DATA_TYPE(BINARY)` when working with image files that are already in a compressed state (i.e. JPEG, TIF) This eliminates any data translation attempts against files that will not need the EBCDIC-to-ASCII translation performed, saving CPU and elapsed time.
- Specify `COMPRESSION_METHOD(STORE)` when working with image files that are already in a compressed state or smaller dataset sizes (i.e. 1KB to 5KB)
  STORE mode performs no compression against datasets, and can reduce CPU utilization and wall clock time when working with datasets that are already compressed or are very small in size.

4 Efficient Use of Transfer Between Platforms

We worked with an international financial services and banking company who needed to provide hundreds of files from their z/OS system to a variety of Open Systems platforms including Windows Server, Linux and AIX on a daily basis. They had been running regular file transfers of these files which was a costly (elapsed and CPU time) and cumbersome step.

Compression support for z/OS means that when compressing a file, approximately 90% of the CPU workload is zIIP eligible and only 10% is processed by the general CP's. Once the initial cost of the special purpose engine has been paid for, more processing can be put on that engine – and the data center operator has less to deal with. With a resource that is able to read and write directly to UNIX file system supported files, those files were written directly to NFS mounted file systems that the Open Systems platforms were able to consume immediately.

INTEROPERABILITY BEST PRACTICES

Movement of data between operating platforms is standard in today's IT infrastructure. Even though the data format (EBCDIC vs. ASCII) may differ, the integrity of the data needs to be consistent. PKZIP and SecureZIP support the interoperability between operating environments, allowing ASCII-based users to extract and process data received from a mainframe or vise versa. As noted above with the File Transfer Best Practices, the following parameters should be reviewed and modified accordingly:

- `DATA_TYPE`
- `DATA_DELIMITER`
- `FILE_TERMINATOR`
- `ZIPPED_DSN or UNZIPPED_DSN`

Summary

Compression can now be performed on the IBM zIIP special purpose engine as well as the zEDC. Compression can be inserted into existing work flow and reduce the overall time it takes to transfer files.

Mainframe datasets when archived in a format like ZIP, can be stored in any storage platform, not restricted to only the mainframe, and without any dependency on hardware. Should you ever need to restore the contents to the mainframe, dynamic allocation will automatically restore the data to its original contents.

To ease the complexity of many files, hardware assisted ZIP compression and encryption provides the ability to aggregate many files into a single file, easing the burden of managing too many objects.

ZIP is like a portable file system, in that it allows files to be compressed and encrypted, utilizing the most efficient software and hardware and consumed on any platform.

Adding compression and encryption is the most straightforward abstracting at the file level, because it does not require any program changes, and does not alter the existing workflow of information; it simply augments it. In the case where high performance is critical, program modification using Virtual Objects to interface with compression and encryption facilities a binary abstraction to achieve a significantly higher level of performance.

Widespread adoption of compression and encryption are possible now on z/OS without needing to increase the MSU capacity of your machine. Instead, you can use assistance from hardware through the zIIP and zEDC for compression, and CPACF and the Crypto Express cards for encryption.

FOR MORE INFORMATION ON IMPROVING DATA CENTER PERFORMANCE IN YOUR ORGANIZATION, CONTACT PKWARE—800.219.7290

PKWARE®

CORPORATE HEADQUARTERS
648 N. Plankinton Ave.
Suite 220
Milwaukee, WI 53203
1.800.219.7290

UK / EMEA
Building 3 Chiswick Park Chiswick High Road,
London W4 5YA
United Kingdom
+44 (0) 208 899 6060