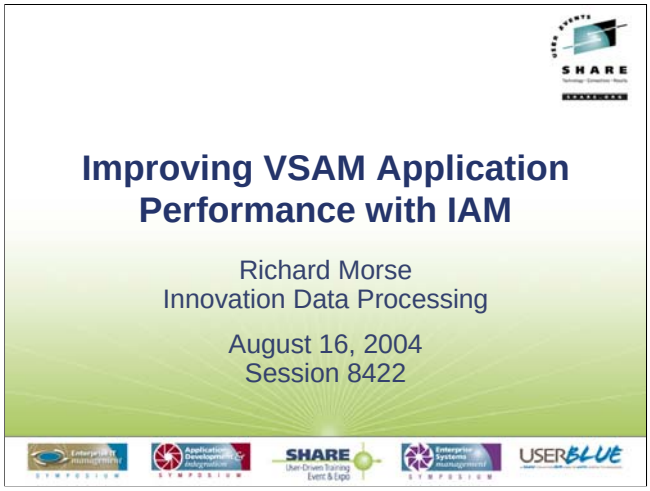
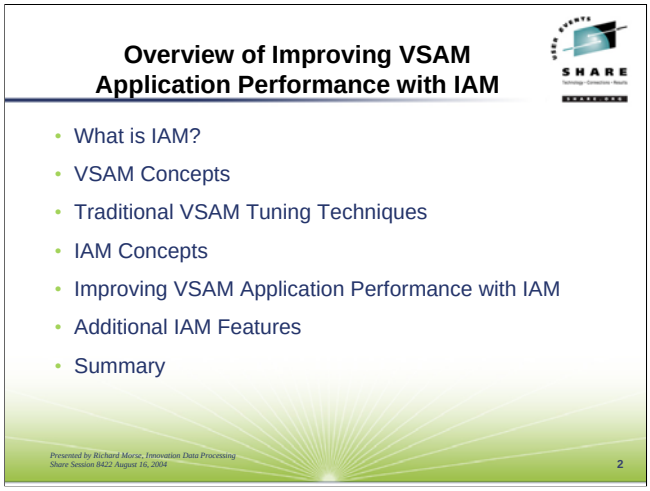
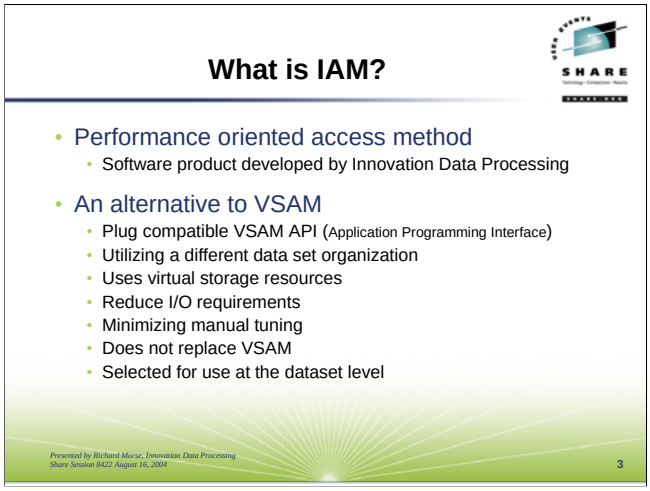


This session presents at the technical concept level, how IAM improves the performance of VSAM applications. VSAM has been in existence for some 30+ years. In spite of the great success of DB2, there still remains vast amounts of data stored on mainframes in VSAM format. The performance of accessing VSAM data remains of high importance in many industries for critical applications. IAM has been in existence almost as long as VSAM, and has provided improved VSAM application performance for many companies. The performance improvement is achieved by reducing physical I/O and processor cycles to access and update application data, resulting in reductions in response times for online transactions, and elapsed times for batch jobs. The following presentation will contrast VSAM and IAM, to demonstrate how this performance improvement is achieved. The information is being presented at a conceptual level, and is not intended to be a comprehensive information source of the functions, features, and facilities of VSAM or IAM.

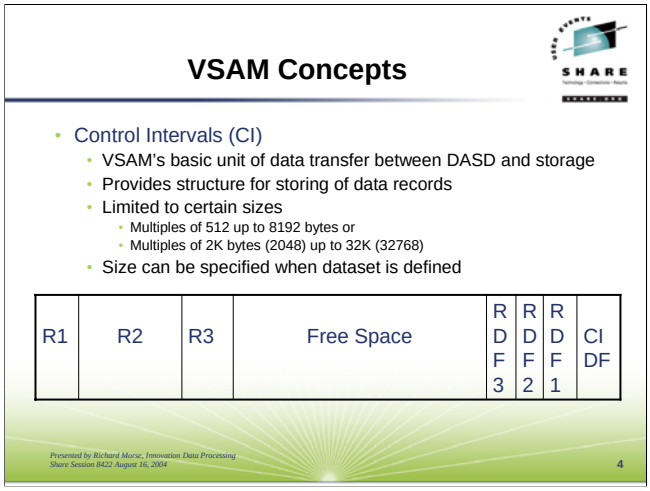


We are going to start with a brief introduction to IAM, so that those who haven't previously heard of IAM have some idea what IAM is. Then we will review some basic VSAM concepts and tuning techniques to understand how VSAM functions and the facilities provided within VSAM to improve application performance. Next we'll present some basic IAM concepts, followed by the functions and features that IAM provides that lead to improvements in VSAM application performance. Subsequently, we'll briefly present some additional interesting features of IAM. We'll wrap up by a summarization of the IAM facilities that improve VSAM application performance.



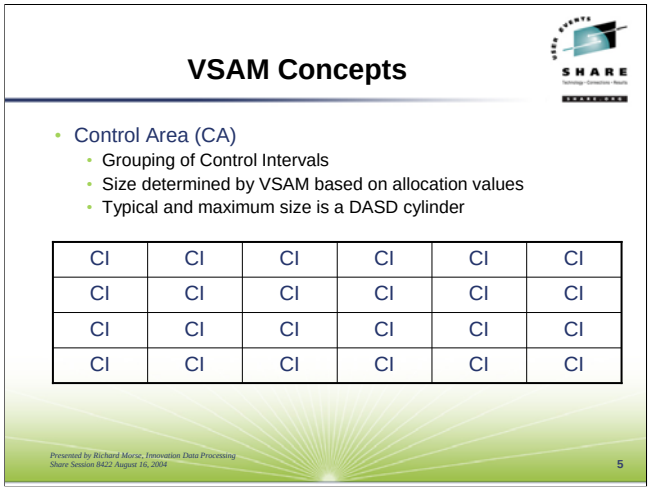
IAM is a performance oriented software product developed by Innovation Data Processing. IAM was initially conceived and developed to enhance the performance of ISAM applications. Over the years, as ISAM use decreased, and VSAM use became more wide spread, it was becoming evident that the facilities and functions provided by IAM also resulted in improvements for VSAM applications. This lead to the development of a compatible VSAM interface to IAM, along with additional performance oriented functions and features. The IAM product is continuing to be enhanced, offering additional functionality and performance enhancements.

IAM is an alternative to using VSAM for many applications, typically providing significant performance improvements. IAM provides a plug compatible VSAM application programming interface (API) that utilizes a different data structure and data set organization, using virtual storage resources to reduce I/O requirements, yielding high performance data access while minimizing the manual tuning effort. IAM can be used without changing application programs, and generally without any or minimal JCL changes. IAM works with many VSAM application software packages, and most utility programs that are used for VSAM datasets. IAM does not replace VSAM, but is an alternative specified when the VSAM file is defined. This selection is done typically by specifying the parameter OWNER($IAM) on the DEFINE CLUSTER request, other selection options are available such as $IAM in the dataset name, or $IAM in the DFSMS storage class or data class name.



The root of the VSAM data structure is a control interval. A VSAM control interval (CI) is the basic unit of data transfer between DASD and virtual storage. The control interval provides VSAM's structure for storing data records. For most other access methods, the basic unit of transfer and data structure is called a block. There are some key differences between a block and a control interval.

First, the data structure within a control interval is different from that used in a typical block. In a data block, if it contains fixed length records, there will be no block control information, just each data record. In a variable length data block, the first four bytes have the BDW, which contains the length of the data block, followed by the data records, each of which is preceded with an RDW containing the length of the record. The RDW is treated as part of the record, and is passed to the application program as a part of the data record. For VSAM, the actual data records start at the beginning of the control interval. The control information about the control interval is maintained at the very end of the control interval, occupying the last four bytes. Those four bytes contain the amount of free space within the control interval, along with the offset from the start of the control interval to the free area. Then, preceding the control interval descriptive information (CIDF) are record description information (RDF), which contains the record length and various flags consisting of three bytes each. The first record in the control interval is described in the RDF immediately preceding the CIDF. This is preceded by the RDF's for subsequent records moving towards the actual data records. There is typically an RDF for each record contained within the control interval, however there are exceptions as indicated by the flags within the RDF. The primary exception is if there is a set of records of equal length, except for a fixed length record RRDS, then there will only be two RDF's for the set, one with the record length, and the other with the count of number of records of the same length. In the VSAM CI structure, the record length is not part of the actual data record, as is typical for non-VSAM variable length data records, and is not passed to the application in the data record. Rather VSAM passes the actual data length in the RPL (Request Parameter List) for the I/O request. The free space within the control interval sits between the end of the last data record, and the control information beginning with the RDF for the last record in the CI.

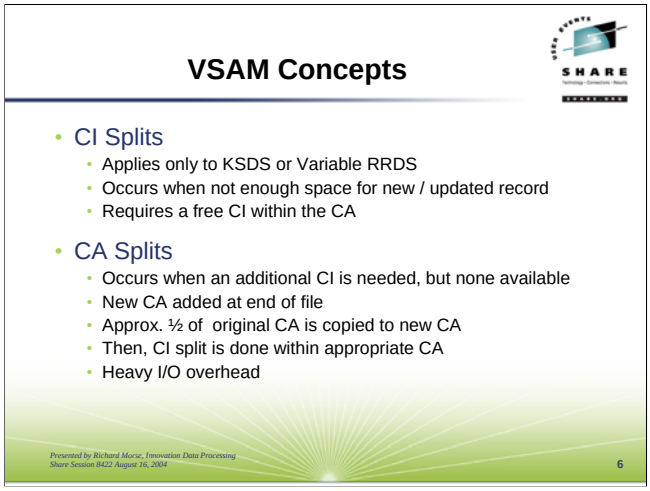


Another difference between a block and a control interval is that control intervals can only have certain specific sizes. Specifically, permitted sizes are 512 bytes, or increments of 512 bytes up to 8192 bytes (8K), or larger amounts in increments of 2048 (2K) bytes up to 32768 (32K) bytes. The control interval size can be specified, or defaulted to, when the VSAM dataset is defined. These permitted CI sizes may not make the most use of a particular DASD track architecture, and certain sizes may be better than others for specific devices.

A third difference between a control interval and a block is that a control interval may consist of multiple physical records on DASD, where as with blocks, the block size is the actual physical record size on the media. While VSAM does generally use a physical record size equal to the control interval size, different sizes may be used to provide for better device utilization.

The next building block in the VSAM data structure is called a Control Area or CA. A control area consists of a set of control intervals. Within any VSAM dataset, the number of control intervals in each control area is fixed. VSAM determines the size of the control area, based on the primary or secondary space allocation. The most common and maximum control area size is equal to one DASD cylinder.

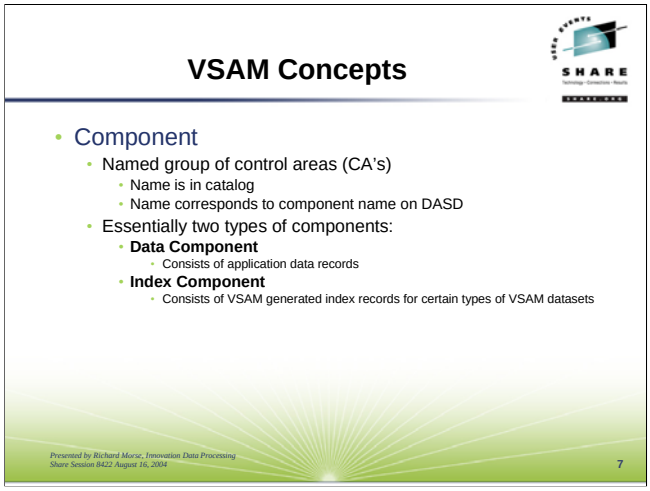
For example, using a 3390 device geometry, and a control interval size of 4096, there can fit 12 control intervals per track times 15 tracks per cylinder, yields 180 control intervals per control area.



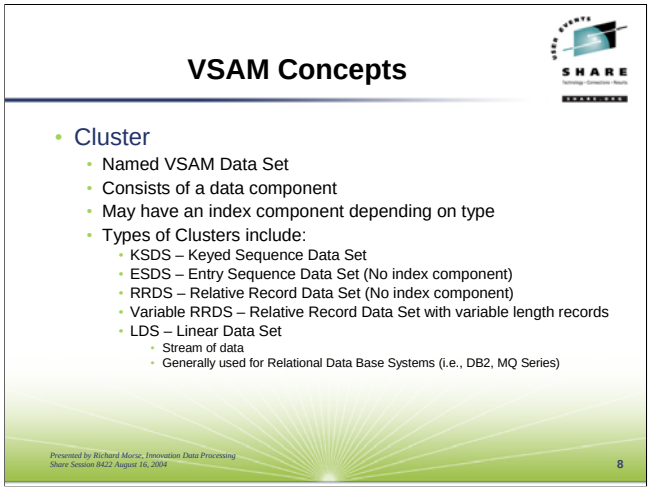
One of the unique features of VSAM is how it handles records being inserted into an indexed file (KSDS or Variable RRDS), or updated records with an increase in length. If there is no room in the CI for the new or updated record, VSAM divides the data in the control interval into a second control interval in an operation referred to as a CI split. For a CI split to occur, there must be a free control interval within the control area. When the CI split occurs, of course the index structure is updated appropriately.

If there are no free CI's within the CA, then VSAM moves some of the data into a new CA, in an operation referred to as a CA split. For a CA split, a new CA is obtained from the end of the data set. Then, a portion of the CI's in the old CA are moved over to the new CA, and the index structure is updated. Then, the CI can be split into a free CI within the CA that the CI being updated is in.

As with any insert strategy for an indexed file, this technique has its advantages and disadvantages. Structurally, it is a nice solution as it leaves a structure that may be able to accommodate substantial additional insert activity without being reorganized. Looking at the disadvantages, included are the heavy I/O requirements particularly to perform CA splits and the resulting structure may result in large portions of unused space within the VSAM dataset.



Moving up to the next level in the VSAM dataset structure is the component. A component is a named group of control areas. This name, which can be specified when the VSAM dataset is defined, is in the catalog structure as well as the name that exists in the DASD VTOC (volume table of contents). There are two types of components, a data component and an index component. All VSAM datasets have a data component, which contains the actual application data records. KSDS and variable RRDS datasets have an index component. The index component consists of VSAM generated index records that provide direct access to specific records. Each component can have a different CI size, different space allocations, and may even reside on different volumes.



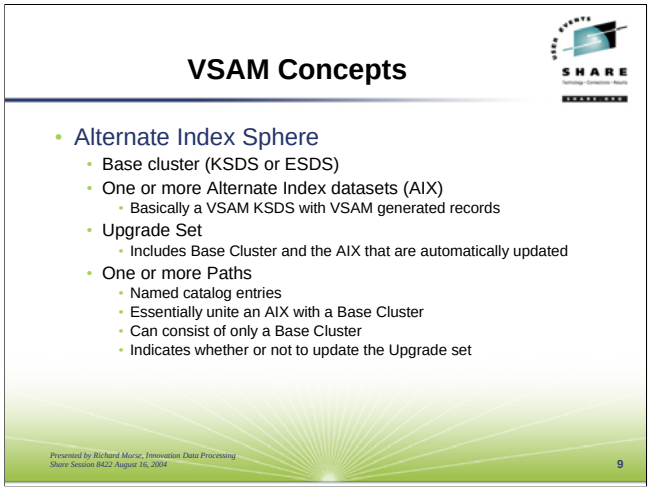
A cluster is the basic VSAM dataset, as this is what is defined and generally what is referenced in JCL. (While the components can be separately processed by name, this is not typical for applications processing.) A cluster is composed of a data component, and if it is a KSDS or Variable RRDS, it will also have an index component. The cluster name exists in the catalog structure, it is not on the VTOC (volume table of contents) on the volumes on which it has components, rather as mentioned previously the component names are what exist in the VTOC.

There are four types of clusters that provide a record oriented interface, and these are the types typically used directly by application programs. Of the four, the type in most prominent use is the KSDS, which stands for keyed sequence data set. A portion of the data record is assigned as the key, which serves as a unique identifier for each record within a KSDS. The uniqueness is critical, as there can be only one record in any KSDS with that particular value. The records are maintained, through the use of the CI/CA structure and the index in ascending key sequence. The KSDS type has the most functionality in that it provides support for variable record lengths, records can be inserted at the front of the file, between existing records, or at the end. Records can also be updated, they can be updated with a length change, and records can be deleted. All that function comes with the costs of maintaining the ascending key sequence through the dataset structure and the index.

The second most used type of VSAM cluster is the ESDS, or entry sequence data set. This is a sequential dataset, and records are maintained in the order that they were added to the dataset. There is no key in the data to form a unique identifier. However, the records in the dataset may be randomly processed, providing the application has a way to store and retrieve the RBA assigned to each record, or has the ability to calculate an RBA. An RBA stands for Relative Byte Address, and represents the offset within the ESDS as to where the record is located. Records can be added to the end of an ESDS, but they can not be inserted between existing records as that would cause a change in the record's RBA, which is its address. Also while records can be updated, they must retain the same length as the originally had because a length change could cause the address (RBA) of subsequent records to change. Records can not be physically deleted from an ESDS, although an application can develop it's own conventions for logically deleting records.

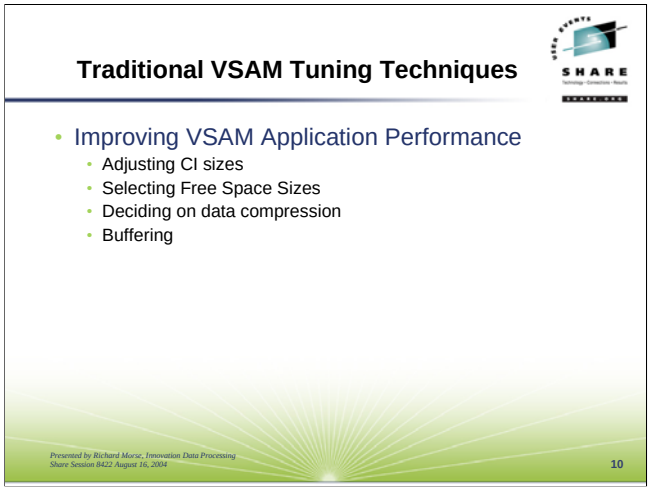
The last two record based clusters are the fixed length record RRDS, or relative record dataset, and the variable length record RRDS. The records in these datasets can be accessed sequentially, or randomly by their relative record number. A fixed length RRDS does not require an index, because by knowing the record length, the CI size, and the relative record number the location of the record can be calculated. An application does not have to use all of the available record slots, although there will be space reserved for each record up to the highest record number stored in the dataset. A fixed length RRDS can also be loaded in random record number sequence. This type of dataset is well suited for using a hashing algorithm from some data value in the record to compute a record number. The variable length RRDS is essentially a KSDS, where the keys are not necessarily within the data record and consist of the relative record number. While similar in function in many ways to a fixed length record RRDS there are some differences. In particular, they can have records of different lengths, updated records can have their length changed, and it must be initially loaded in ascending relative record number sequence.

The last type of cluster is an LDS, or linear data set. This is basically a stream of data that is stored and retrieved in 4K (4096) byte increments. This type of cluster is typically used by products such as DB2 and MQ Series. IAM does not provide support for this type of dataset.

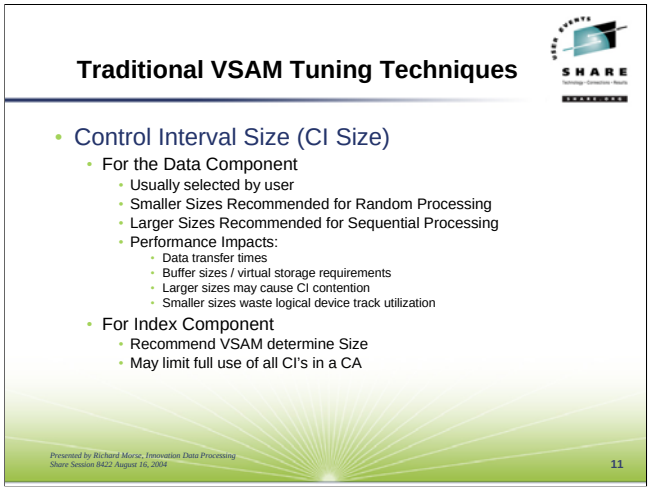


A KSDS or ESDS type cluster structure can built into a sphere by using alternate index datasets. The alternate index provides a way to retrieve records in an alternate sequence, or find records randomly with a key value different from the base key value. Alternate index datasets are basically a VSAM KSDS cluster, but contain records in a VSAM specified format. The key of each record is an alternate key value from the data record in the base cluster. The data within the alternate index record consists of some control information, the alternate key, and then either the key of the record in the KSDS, or the RBA of the record if it is in an ESDS. An interesting attribute of an alternate index is that it may, if so specified when it is defined, have a single alternate key index to multiple records in the same base cluster. Another attribute of an alternate index, called upgrade, indicates whether or not it is to be automatically updated when records are inserted, deleted, or updated in the base cluster resulting in changes being required to the alternate index. This automatic upgrade will insure that the alternate index is always synchronized with the data records in the base cluster.

A base cluster can have multiple alternate index datasets associated with it. To access the data in a base cluster through an alternate index, a PATH must be defined. For VSAM, a PATH is just a catalog entry, somewhat similar to an alias entry, except that when the PATH is opened, VSAM processes the base cluster through the alternate index that the PATH indicates. Typically, each alternate index will have at least one PATH related to it, although it could have multiple paths. A PATH entry also has an attribute that indicates when it is opened for update processing, whether or not VSAM should open and update all of the alternate index datasets in the upgrade set. The upgrade set consists of all of the alternate index datasets associated with a particular base cluster that have the upgrade attribute.



Now that we've seen an overview of what a VSAM data set is, we are going to look at some of the basic parameters and VSAM facilities that impact overall application performance. This includes choosing control interval sizes, free space values, whether or not to use data compression, and selecting proper buffering. Of these, the first three are specified when a VSAM data set is defined. The last one, buffering, can be adjusted whenever the VSAM dataset is used.

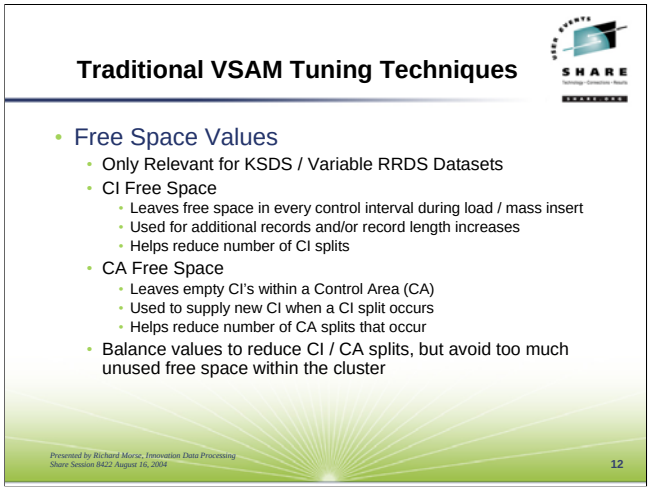


The control interval size is quite important, as it reflects the amount of data transferred per each I/O operation. If one is not specified, VSAM defaults to fairly large size. That is good for device utilization, and sequential processing. However, if the dataset is going to be used for random processing, or online processing, then smaller control interval sizes would likely be better.

For the data component, it is best to specify the size desired, based upon the type of processing that is being performed on the data set. Smaller sizes, such as 4K or perhaps lower, are generally recommended for random processing. The smaller sizes are also beneficial if there is a fair amount of online update activity. If all of the processing is sequential processing then larger sizes are recommended.

Some of the considerations when selecting a CI size include factors like data transfer times, buffer sizes and virtual storage use, control interval contention particularly when using LSR buffering, and DASD space utilization. Larger CI sizes will cause longer data transfer times, larger buffer sizes, with increased virtual storage requirements. Additionally, for data sets that have a fair amount of update activity, particularly online updates, large CI sizes can cause delays due to CI's being locked while a request is holding it for update purposes. On the plus side, larger CI sizes (up to a point) will improve DASD space utilization, and may also result in better utilization of free space, resulting in fewer CI splits.

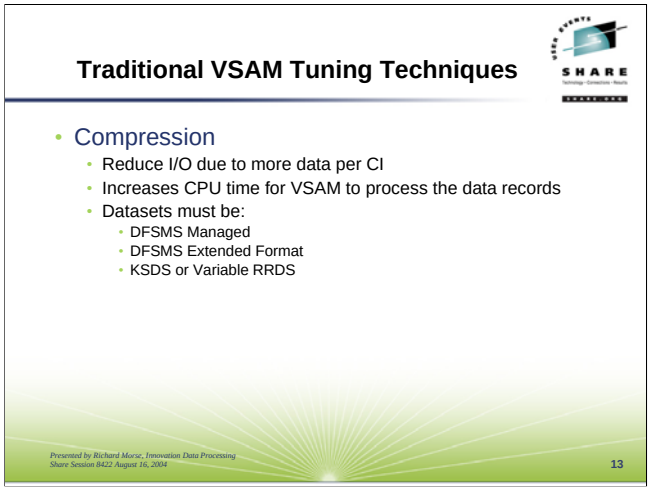
For the index component, it is generally recommended that VSAM determine the size. The index component CI size must be large enough to hold what is called the sequence set for each CA. The sequence set CI contains the index entries for each data component CI within the CA. With VSAM determining the number of CI's per CA, and being able to calculate the expected index entry size, VSAM can best determine the size needed for the index component CI. A size too small will result in VSAM not being able to make use of all of the data component CI's within a CA, and too large a size can hinder performance due to unnecessary longer data transfer times. Unless you have determined that VSAM is calculating too small of an index component CI size, let VSAM determine the proper size.



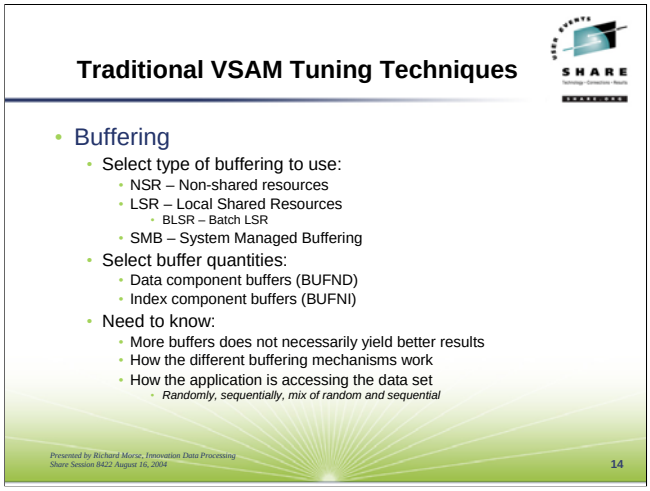
The next values are the CI and CA free space values, which are specified when the file is defined. These are important for good performance because appropriate values may help significantly reduce the high overhead of CI / CA splits. On the other hand, don't make the values too large such that they cause a lot of unused space within the VSAM data set. The free space values are specified as a percentage of the CI or CA to keep free as data is being loaded into the data set, and in certain mass insert situations.

CI free space leaves approximately the specified percentage of bytes unused in each data component control interval. This leaves space to accommodate records being inserted or records being updated with a longer length, which helps reduce the number of CI splits that may be necessary. VSAM will always put at least one data record in each CI other than the free CI's for CA free space.

CA free space leaves empty CI's within each CA as the file is being loaded. These CI's are available for use as the new CI when a CI split occurs. This is an important value because it is desirable to avoid CA splits when possible due to performance, however one doesn't want to waste space either. As an example, in a VSAM data set with a 4K CI size, and a one cylinder CA size, VSAM can fit 180 CI's for that CA. Specifying 10% free space will leave 18 CI's within each CA empty.

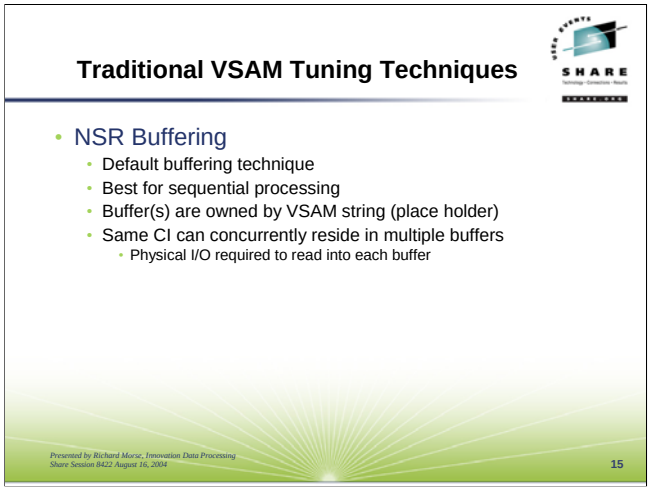


Data compression with VSAM can aid performance by reducing I/O activity because the result should be more data records in each CI. This will also aid in reducing DASD space utilization. The cost of data compression is an increase in CPU time, particularly when loading, adding, and updating data records in the file. If there are available CPU cycles, then using data compression may be quite beneficial if your data sets meet the requirements. To use the DFSMS data compression function on VSAM data sets, it must be for a KSDS or a variable RRDS, and must reside on DFSMS managed volumes in the DFSMS Extended Format.



VSAM offers a few different buffering techniques for users to decide on what will give the best performance for particular data sets and applications. Amongst the choices available for most applications are non-shared resources (NSR), local shared resources (LSR) or System Managed Buffering (SMB). Once a technique is decided on, then one also needs to decide on the number of buffers needed. This becomes complicated for KSDS and Variable RRDS files, because there are two values, data buffers (BUFND) and index buffers (BUFNI).

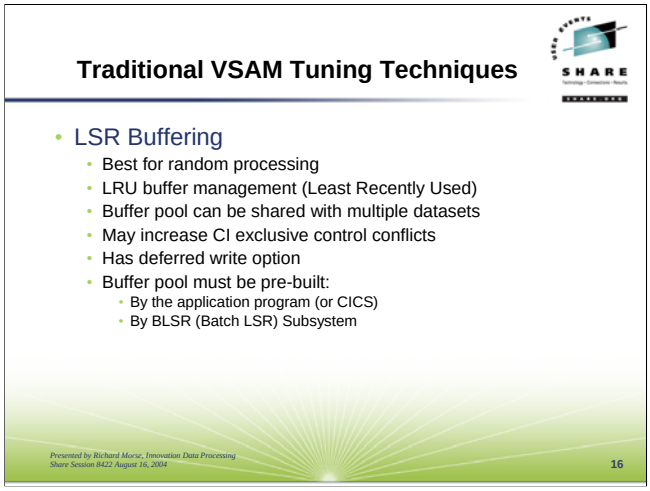
So, to choose in a way that will help performance, one needs an understanding of how each buffering technique works, how the application is accessing the data, and then come up with both a number of data buffers and a number of index buffers. One other thing to watch out for is that providing too many buffers may result in additional CPU cycles being used by VSAM just in buffer management for buffers are that not effectively being used. So, some care needs to be taken when selecting the number of buffers.



VSAM's default and original buffering technique is called NSR buffering meaning non-shared resources. That means that the buffers and related control blocks are not being shared with other VSAM data sets being processed within the job. The default number of buffers that VSAM will use is based on what it determines to be the minimum number of buffers it will need to process the data set. This involves knowing a bit about VSAM internal processing, in that the minimum number is based on the number of strings. A string, or a place holder, is required for each concurrently active I/O request. Additionally, for many I/O requests, the string may be held even after the I/O request has completed. For example, a string is held for sequential processing, for update processing, for locate mode processing, and for other random requests where positioning is requested to be maintained. A string basically keeps track of where within the file the last record came from, so VSAM will know how to get to the next record. Each string effectively owns the buffers that it has been using.

Presuming a KSDS is being processed, VSAM will need a data component buffer and an index component buffer for each string, plus one additional data component buffer to handle CI / CA splits. Additional data component buffers are beneficial for sequential processing for VSAM to perform read ahead processing and for sequential updating to delay writing until there are multiple CI's that need to be written. Additional data component buffers are not used for random processing. Additional index component buffers are beneficial for random processing, and will hold the index CI's for the higher levels of the index that are above the sequence set.

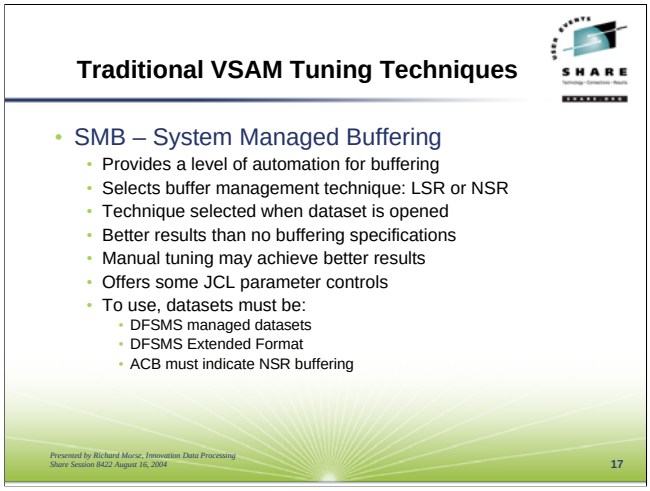
An interesting note with NSR buffering is that the same CI may be in multiple buffers at the same time, however only one of them can have the CI for update processing.



LSR buffering stands for local shared resources. That means that buffers, buffer control blocks, and strings can be shared between multiple VSAM data sets that are assigned to the same LSR buffer pool within an address space. Within each LSR pool can be a variety of different buffer sizes, in different quantities. There can be multiple LSR buffer pools within any particular address space. An LSR buffer pool must be pre-built by the application prior to opening any VSAM data set that is going to use LSR buffering. CICS will take care of this itself, but other applications will need to build a pool. For batch applications using VSAM datasets, they can go through the BATCH LSR subsystem which will build a buffer pool for the data set, however using that function does require JCL changes.

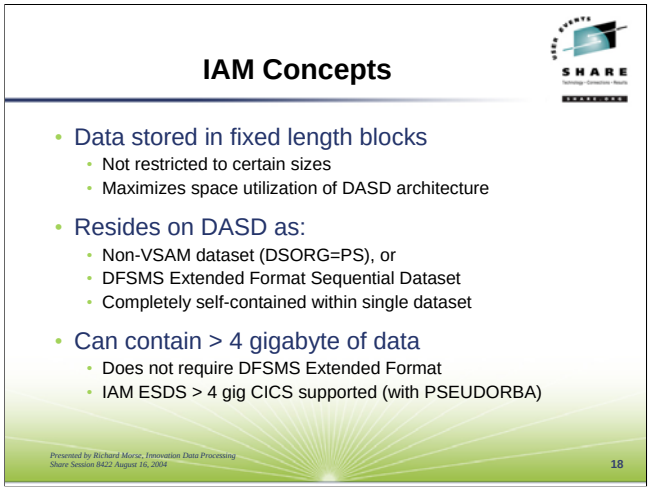
LSR buffering uses a different buffer management technique than NSR buffering, which is that it manages buffers on a LRU (least recently used) basis. This means that control intervals that are frequently referenced will tend to stay in the buffer pool, where as buffers containing control intervals that haven't been referenced in a while can be reused for a different control interval. This type of buffer management generally is good when processing the data records randomly. LSR buffering does not perform read ahead, or read or write multiple CI's per an I/O, so it will not provide much benefit to jobs that process the files sequentially.

Unlike NSR processing, there will be only one copy of any particular CI in the LSR pool at a time. This can result in increased CI exclusive control conflicts, which result in delays. Once a string has accessed a CI in the buffer pool for update processing, it will not be used by other strings until the update is complete. So, even if some other requestor wanted a different record, that requestor could not retrieve the record if it was in a CI that was locked for update processing.



System Managed buffering provides a level of automation in buffering for DFSMS managed data sets that are in extended format. SMB can be specified either via JCL parameters on the DD card for the VSAM dataset, or in the Data Class to which the dataset belongs. SMB will choose an appropriate buffer management technique, along with an appropriate number of buffers based on either direct specification of a buffer management technique, or based on the options specified in the ACB MACRF field for the dataset being opened. Except for file load processing, SMB basically will decide at open time to use either NSR buffering or LSR buffering for the dataset, and compute a number of buffers (data and index) to be used.

In some limited testing, I found that the use of SMB has resulted in noticeably better performance than if no buffering had been specified. Better performance could still be achieved with manual tuning. The primary exception was with file load processing, particularly for VSAM files defined with the SPEED option. SMB performed much better than even manual buffer specifications for file loading.

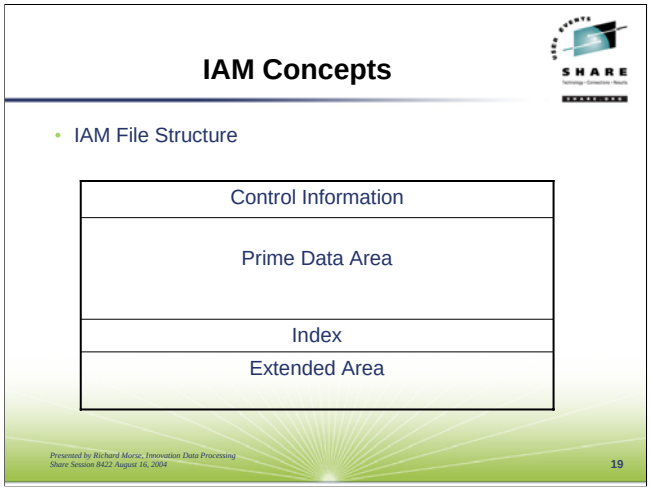


Moving on to IAM, as stated at the previously, IAM uses a different data structure and dataset organization. We'll start with the basic unit of data transfer for an IAM file, which is a fixed length block. One might wonder isn't that similar to VSAM's fixed length control intervals? It is, but only in the concept of using fixed length blocks. The organization of data is completely different, as are many of the other attributes of an IAM block. First, the size of an IAM block is not restricted to an arbitrary set of sizes as VSAM is, which means that IAM can use block sizes that will allow full track utilization on the device geometry on which the data set resides. Therefore depending on data record lengths and CI sizes, in many circumstances IAM can store more data on a track then VSAM will.

The next difference is data record structure within the block. Because the blocks are fixed in length, there is no block descriptor word (BDW). The records start at the beginning of the block are all prefixed with a modified 4-byte RDW, record descriptor word. The RDW is modified for IAM to handle data compressed records, and spanned records. The application program will never see this RDW, just as with VSAM it doesn't see the RDF. Free space is at the end of the block, immediately after the last data record.

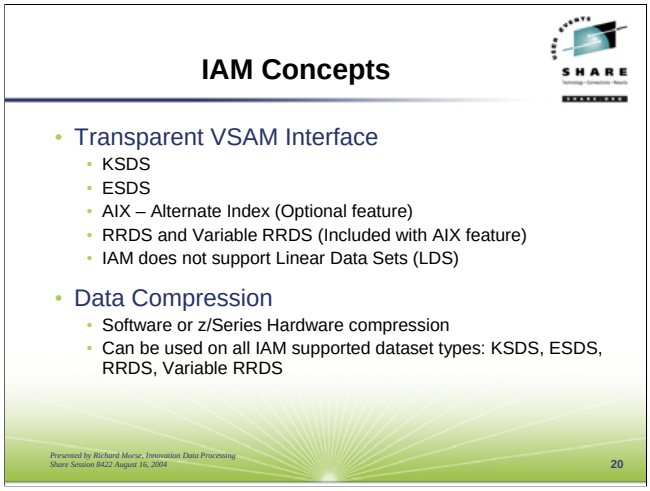
IAM files reside on DASD as non-VSAM datasets, typically with a DSORG of PS (physical sequential), or optionally as DFSMS Extended Format sequential datasets. This means that the cluster name specified when defining an IAM dataset, is the name and only name of the dataset that is in the catalog structure as a non-VSAM dataset, and on the DASD VTOC. All of the descriptive information about the IAM dataset, the user data records, and the index is kept within the single IAM dataset itself. IAM datasets do not have control areas or components. Rather, it is a collection of fixed length blocks containing internal file description information and user data.

One interesting note is that due to this structure, IAM datasets have never had a maximum data limitation of 4 gigabytes. So, IAM datasets can contain over 4 gigabytes of user data without having to be DFSMS managed or in the DFSMS Extended format. IAM ESDS exceeding 4 gigabytes can also be used under CICS with the special PSEUDORBA option for an IAM dataset, when an application is not dependent on the RBA calculations and values as set by VSAM.



Within the IAM dataset itself are basically four different areas. The first area and first blocks within an IAM dataset contain the file description information, things like record length, key size, key length, and information on the file structure and index, such as where within the file the other areas are and their respective sizes. Following this descriptive information is what IAM calls the prime data area. These are blocks containing application data records. The size of this area is based on the amount of data that is initially loaded into the IAM dataset, plus any embedded free space. At the end of the prime data area is the index to the prime data area. This index is read into virtual storage when the IAM dataset is opened, so there is no index I/O to retrieve records once the IAM dataset has been opened. This prime index structure is never updated, until it is rewritten when the IAM file is reorganized.

After the index area there may be an extended area. The extended area is used to contain records that have been added or records that expanded in size for which there was no room within the free space in the Prime Data Area. The extended area has a dynamic structure, which include an area for the some blocks describing the usage of the subsequent blocks, and then the extended prime blocks and the extended overflow blocks which can be intermixed. Extended prime blocks are used to contain records that are added to the logical end of the dataset, that is with keys higher than the high key initially loaded into the dataset. This is similar to the prime data blocks, and is indexed by the high key in each block. The extended overflow blocks contain data records that have been inserted into the file or expanded, and did not fit within the targeted prime block. The extended overflow blocks are indexed by record. This area will be described in more detail later.

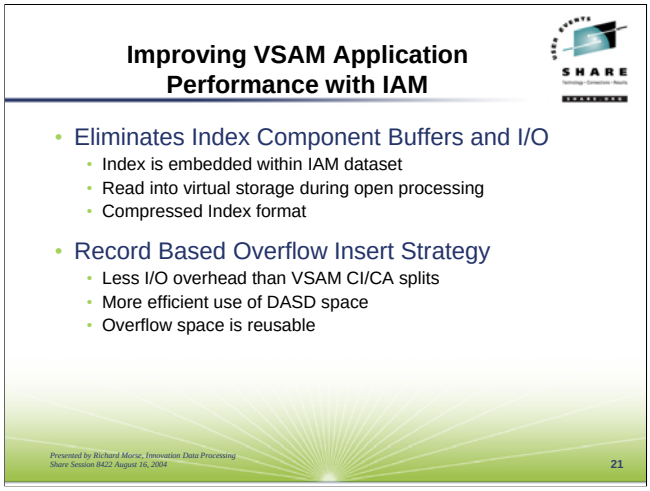


IAM is different than VSAM, but provides a transparent VSAM application programming interface. This means IAM can be used without changing any application programs. In most circumstances IAM datasets can be used without the need to change any JCL. Some changes may be necessary for utility type programs that are involved in data set management, backup, and recovery, but very rarely for JCL executing application programs.

Basic IAM provides support for KSDS and ESDS type of datasets. With the optional AIX feature, IAM provides additional support for VSAM alternate index datasets, RRDS, and Variable RRDS datasets. IAM does not provide support for Linear Data Sets (LDS). One could ask how does IAM provide support for VSAM ESDS with such a different dataset structure? IAM supports VSAM ESDS by generating the RBA (relative byte address) of a record identical to the address that VSAM would generate, and save that in the IAM index structure and in the front of the data record itself, so it can find the data records. IAM ESDS files can exceed 4 gigabytes with a 4-byte RBA by using the PSEUDORBA option, where the RBA values returned will have different values than what VSAM would have used. Many applications can make use of the PSEUDORBA feature, although a few can not. IAM will also support the extended format RBA for ESDS files, without having to be a DFSMS Extended Format dataset.

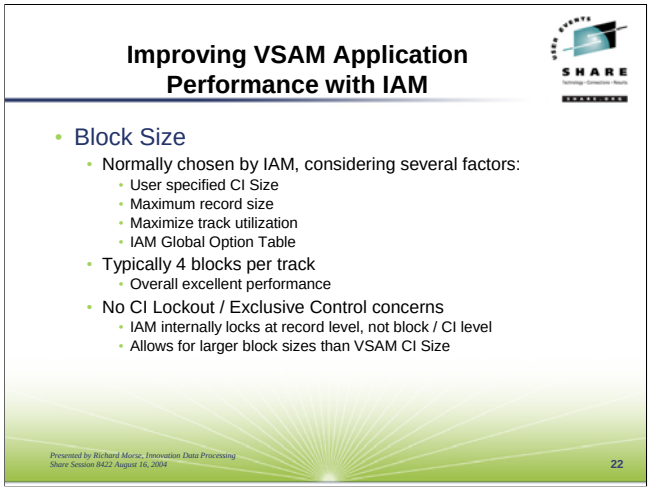
The IAM AIX support provides support for base clusters with alternate index datasets, the alternate index datasets, and the associated paths. No changes are necessary to application programs or CICS to use the IAM AIX functions. The IAM alternate index datasets can be built with the IDCAMS BLDINDEX command, as well as other programs and program products that build alternate index datasets. One difference with IAM AIX support is that the IAM paths are actual one-track datasets, as compared with VSAM paths being only catalog entries. Some utility functions may require modifications to provide identical function for IAM AIX datasets. In particular, most DASD management, backup, and recovery products do not recognize the relationship between the IAM alternate index datasets, paths, and their related base clusters.

IAM also has a data compression feature. Either an IAM provided software compression algorithm can be used, or the z/Series hardware instruction can be used. The data compression can be used on any type of VSAM dataset supported by IAM, including ESDS and RRDS datasets. The data compression function will be described in more detail later.



Having looked at some of the IAM basics, we'll now take a look at some of the ways that IAM can improve VSAM application performance. As stated in the beginning introduction of IAM, IAM will use virtual storage to reduce physical I/O's. The first thing that IAM will do is whenever an IAM dataset is opened, the index for the dataset will be read into virtual storage. To minimize storage use, the index is generally in a compressed format. This virtual storage is either part of the job's region, in which case it is all in 31-bit addressable storage, or it can be contained in a data space. Depending on the size of the file, this may take a little bit of time, but for most files the overhead is not noticeable. With the index residing in virtual storage, IAM never has to perform I/O operations to read the index in response to application I/O requests. This means that IAM can locate and read in any record within the dataset with at most one physical I/O. IAM does not have any index component buffers to be tuned. This function alone can result in significant I/O savings when using IAM instead of IAM.

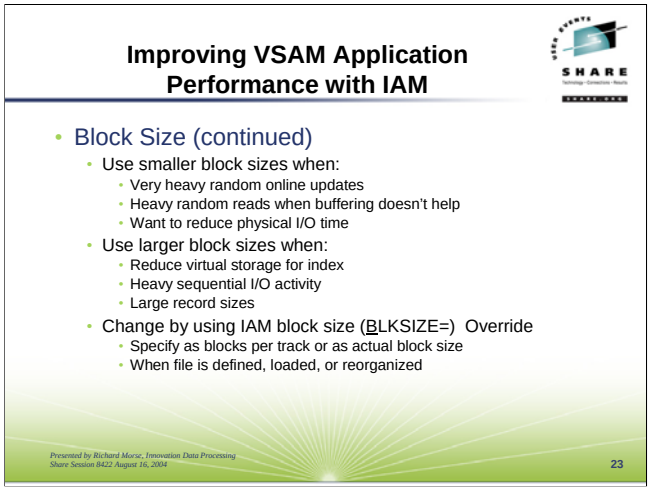
IAM utilizes a record based overflow area to accommodate expanded records that no longer fit within the block they are in, and for inserted records that will not fit in the block in which they would otherwise be placed. When a record has to be placed in an overflow block, or perhaps has to be moved to an overflow block with more room, IAM will find an overflow block with sufficient space, and write the record in that overflow block. IAM maintains in virtual storage a list of overflow blocks that have free space. If there is inadequate space available within the existing extended overflow blocks, then a new overflow block will be obtained from the extended area of the IAM dataset. If there is no room left for additional blocks, IAM will attempt to obtain an additional extent to acquire more DASD space for the extended area. Once added to an overflow block, an index entry is added in virtual storage for the overflow record. Because there is no relation by key of the records within any particular overflow block, any record can be placed in any overflow block regardless of it's key value. Additionally, when a record is deleted from an overflow block, the space is immediately available for reuse for any record that needs overflow space, regardless of the key.



This technique has it's advantages and disadvantages, just as the VSAM CI/CA split technique has. Amongst the advantages of the IAM technique are reduced physical I/O to perform the overflow processing than CI/CA split processing, reduced CPU time to add records to the file, more efficient use of DASD space and retaining the no more than one physical I/O to read any requested record. The disadvantages of this technique are the increase in virtual storage required, potential for long open times when file has very large number of records in the overflow area, and the possibility of excessive I/O when sequentially reading the dataset. For most VSAM applications, they do realize noticeable performance benefits of the IAM overflow technique and find that the advantages exceed the disadvantages and realize performance improvements. The disadvantages may be minimized by increasing the CI free space value, or more frequent reorganization.

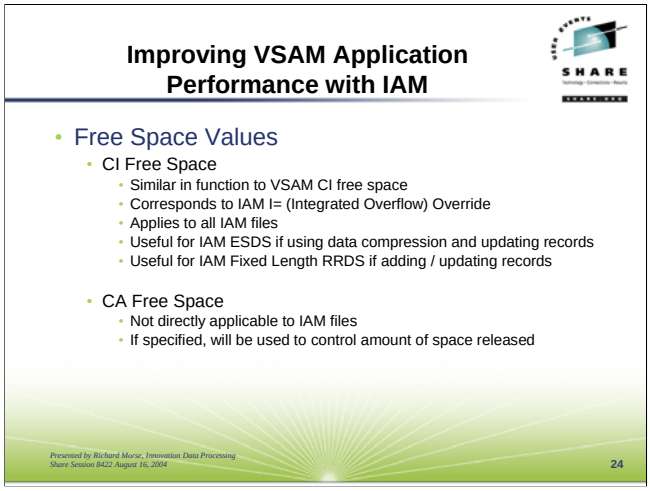
IAM will select a block size for a dataset based on a variety of criteria that includes the user specified CI size if any, the maximum record size, the type of device the dataset resides on, will attempt to maximize track utilization on target device, and can be influenced by changes to the installation options. IAM will generally choose the maximum block size it can that will result in four blocks per track. Depending on the record size and user CI size, IAM may choose a larger block size of three or two blocks per track. For the vast majority of datasets converted to IAM, the IAM selected block size provides excellent performance and rarely needs to be modified. What this means is that users seldom have to be concerned about having to revise the IAM determined block size, either for DASD capacity or performance.

The block size IAM selects are usually larger than the CI size that VSAM is using, particularly if the user had specified a CI size. One of the reasons is that IAM does record level locking within the region, rather than CI level locking. A CI lockout occurs when a requestor is updating a record within a control interval, and another requester may want a different record in the same CI, but is delayed until the first update request is satisfied. IAM locks out on the record basis, so only any concurrent updates to the same record will be delayed by IAM, not any updates for other records that may happen to reside within the same data block. With IAM, users don't have be forced into smaller block sizes due to the CI exclusive control contention.



While it is not frequent, there are some circumstances where using a different block size can improve the performance. For example, you might want to try using a smaller block size if you have a file which has very heavy online updates, or has very heavy random read activity where the buffering is not significantly reducing the physical I/O's. For the latter case, you might want to try using the IAM Dynamic Tabling feature though before changing the block size. You might want to increase the block sizes when there is large virtual storage requirements for the index to the prime data area, when there is heavy sequential I/O activity, or for larger record sizes.

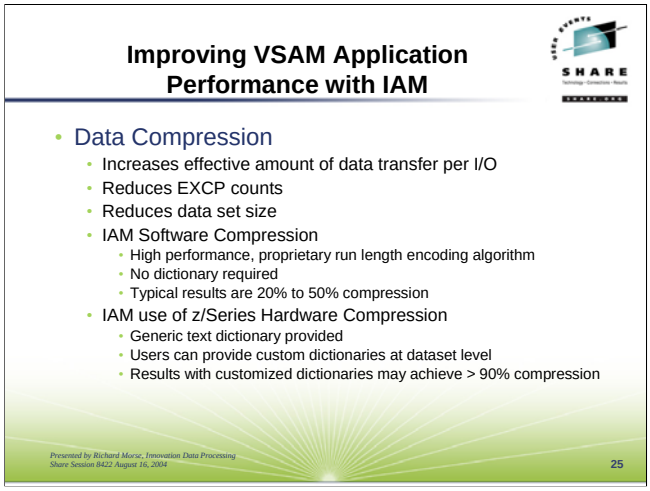
To use a larger block size, either increase the CI size specified in the file definition, or use the IAM CREATE B= override. If you want to reduce the block size to something lower, you can specify a smaller CI size (if it is larger than the 4 blocks per track block size), otherwise you will need to use the IAM CREATE B= override. While you can supply a specific block size, specification of a blocking factor between two and fifteen is recommended, so that you will still achieve the maximum track utilization for the device for whatever size you need.



The VSAM CI free space value is used the same way in IAM as it is in VSAM. It specifies, as a percentage, the amount of space to be left empty in each block in the IAM dataset, as it is being loaded. Just as it can help to avoid CI/CA splits in VSAM, it will help avoid having to put records into the IAM extended overflow area. Also, because data compression is generally defaulted to at most installations for IAM datasets, some free space may be useful if records are being updated in data compressed files. While applications themselves may not increase the record length, when an updated record is recompressed, it could be equal, shorter or longer in length than it was originally. Leaving a little bit of free space for a data compressed file that will be updated is recommended.

The CI free space can also be specified via the IAM I= (Integrated Overflow) override. This may be necessary for ESDS or RRDS type of datasets. While IDCAMS will accept the FREESPACE parameter when defining those types of clusters, the information is ignored. For ESDS files, it is handy only for files that will be updated and have data compressed records. For RRDS, if records are being updated or later added to the file, the free space will also be helpful. For each of those cluster types, and installation default value for free space can be set.

The CA free space value is not directly applicable to IAM files, because IAM does not have control areas. However, if specified, it can reduce the amount of space released when an IAM file is loaded. Normally, IAM will release all unused space at the end of a file load. However, if insert or update activity is expected, then specifying a value for CA free space will tell IAM not release all of the unused space, but to retain a portion of it so that the IAM will not have to immediately take an extent when a record is added.

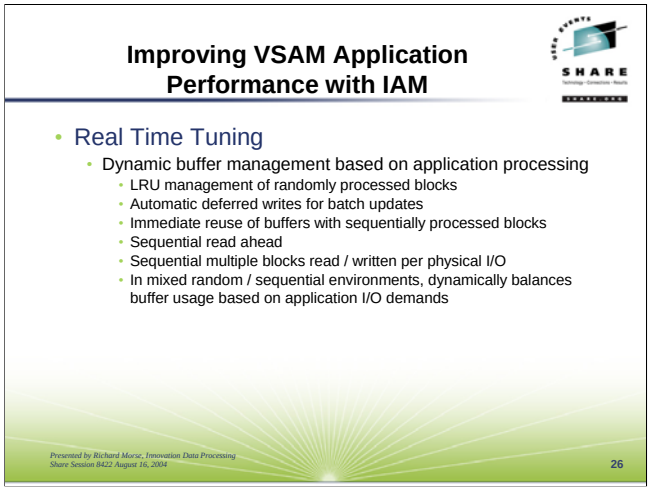


As the IAM product is shipped, IAM defaults to using software data compression on any IAM dataset that is allocated with 5 cylinders (75 tracks) or more. Data compression provides many benefits, by increasing the effective amount of data within a data block, there is more data transferred per physical I/O with no additional I/O cost. That benefit results in reducing the I/O activity (EXCP counts) to IAM datasets. It also reduces the DASD space requirements of the IAM datasets, which also results in a smaller index meaning less virtual storage use. Most installations retain that default, and find the compression to be quite beneficial. Data compression can be used on any type of IAM file, KSDS, ESDS, RRDS, Variable RRDS, or AIX. The down side of data compression is that is uses processor cycles to compress and decompress the data. For many datasets, even with data compression, IAM still uses less CPU time than what was used for an uncompressed VSAM dataset. However, as the record sizes increase, so does the data compression overhead. Most of the CPU time used is to compress the data, decompression is usually much less. Because of the benefits of data compression, we generally recommend its use. However, if CPU time is a constraint, then turning it off particularly for files with very long records, will provide you with CPU time savings.

IAM has two different methods of compression. The first is a proprietary software compression algorithm, that compresses data records by eliminating repeating values. For example, a string of blanks can be reduced to one byte. There is no compression dictionary used by this algorithm, and it was designed to provide a some data compression with a low CPU time cost. Typical results are 20% to 50% compression, and with some files even better compression may be achievable.

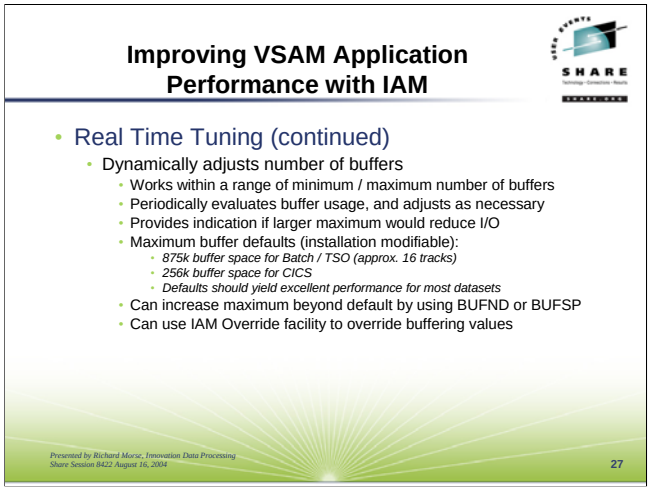
With the performance enhancements provided by IBM with the z/Series processors on the compression instruction, IAM implemented a capability to use that instruction. Currently, IAM provides a generic dictionary that primarily is for text oriented data, such as name and address files. Users can also create their own customized dictionary based on the actual data in the files for use by IAM. Results with customized dictionaries have generally been excellent, with frequently greater than 90% compression being achieved. The CPU time used is similar to the CPU time used by the IAM software compression. The IAM datasets using IBM hardware compression do not have to be DFSMS managed or in DFSMS Extended Format.

Data compression provides a number of performance benefits that reduce resource requirements to access data, so unless you are really tight on CPU cycles, it's use is highly recommended.



IAM's Real Time Tuning feature is one of the major contributors to improving VSAM application performance. IAM dynamically monitors the I/O activity to an IAM dataset, and adjusts various parameters dynamically, including buffer management technique and the number of buffers being used for the dataset. This feature simplifies and reduces the effort involved in tuning VSAM applications using IAM files. Over the next few slides we'll look at the functions of IAM's Real Time Tuning, and how it improves VSAM application performance.

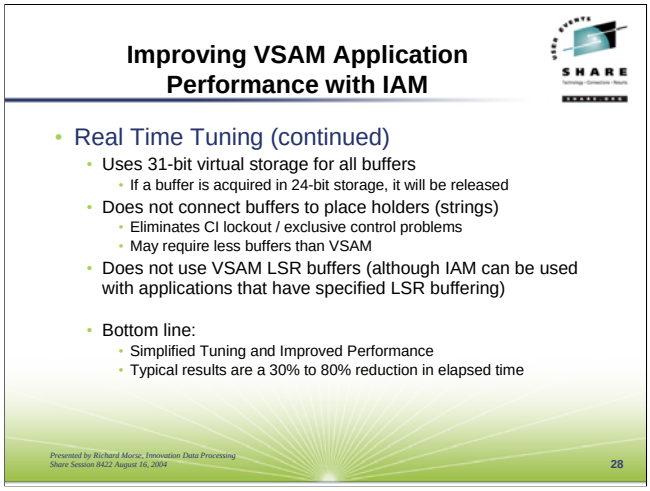
The first function of Real Time Tuning if dynamic buffer management. IAM utilizes a buffer pool for each opened IAM dataset. Those buffers are managed based on how the application program is accessing the data within those buffers. IAM utilizes a least recently used (LRU) buffer management for randomly accessed data blocks. IAM also automatically defers writes for batch updates on datasets with share options of 1 or 2. For sequential processing, IAM will read ahead multiple blocks, read multiple blocks per physical I/O, and write multiple updated blocks per physical I/O. As the sequential process completes it's use of a buffer, it is made available for immediate reuse. Additionally, when performing a sequential multi-block read, IAM will attempt to reuse the same buffers it used for the prior sequential I/O. For applications performing a mix of random and sequential I/O, IAM dynamically balances the number of buffers used for each type of buffer management based upon the actual application use to best meet the application's current requirements. IAM also recognizes whether or not it is running under a batch job or CICS online system, and for example is more aggressive with sequential buffer management for batch jobs than for CICS online systems, because CICS online systems typically have a lot more concurrently active I/O requests than a batch application.



Another function of IAM's Real Time Tuning is that it will dynamically adjust the number of buffers that it is using, on a dataset by dataset basis. IAM works within a default or user specified range of number of buffers, referred to as MINBUFNO and MAXBUFNO. IAM does this by monitoring the applications use of buffers, with internal statistics being kept as if IAM actually had more buffers than it presently has. On a periodic basis based on actual file I/O, IAM will review the activity statistics and increase or decrease the number of buffers one at a time. IAM examines increasing buffers more frequently than decreasing buffers to avoid being in a state of constantly releasing and acquiring of buffers. The intent is to achieve as stable of a state as possible with buffers. IAM considers the quantity and frequency of physical I/O requests, and the nature of being for random or sequential activity.

IAM provides an indication if more buffers would have helped improve performance, that is reduce I/O. This indication is made known via the IAMINFO run time report, which is also available from the optional IAM SMF records.

As IAM is shipped, IAM will for batch jobs is to set MAXBUFNO to the number of buffers for any particular dataset that will fit in 875K of virtual storage. This value allows IAM to obtain a number of buffers that slightly exceeds a DASD cylinder of data blocks. For IAM datasets opened under CICS, a lower value of 256K is used because there are frequently many more IAM files opened under a CICS region than any particular batch application. The default values can be changed at an installation level through the IAM Global Options Table. The values can be increased on an application by application basis through using the VSAM BUFND or BUFSP parameters. Both the minimum and maximum can be explicitly specified if desired through the use of the IAM override facility.

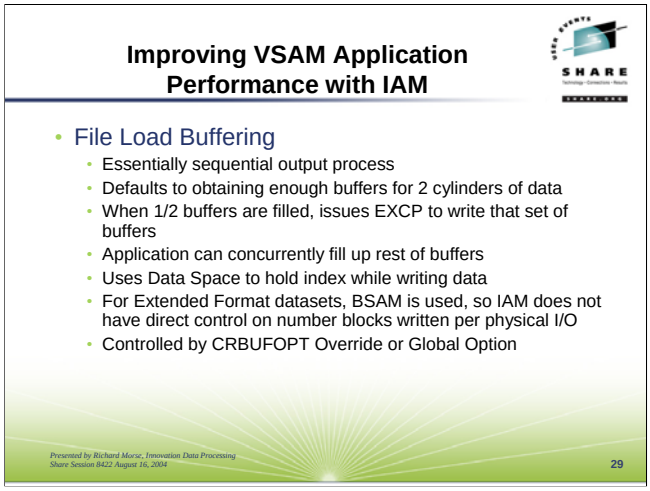


IAM requests that all buffers be obtained in above the line storage. When additional buffers are acquired after OPEN processing, if the storage was actually from below the line storage, IAM will release the storage for the buffer. This is done to help prevent a potential severe storage shortage.

IAM also does not lock buffers to strings (place holders) as VSAM does. This eliminates the CI lockout / exclusive control problems that may be experienced when using VSAM. Instead, IAM uses record level locking. This also provides IAM the ability to use less buffers than VSAM because it is not required to have a buffer for each string.

IAM can be used by applications that specify LSR buffering, however IAM will not use any of the buffers in the VSAM LSR buffer pool.

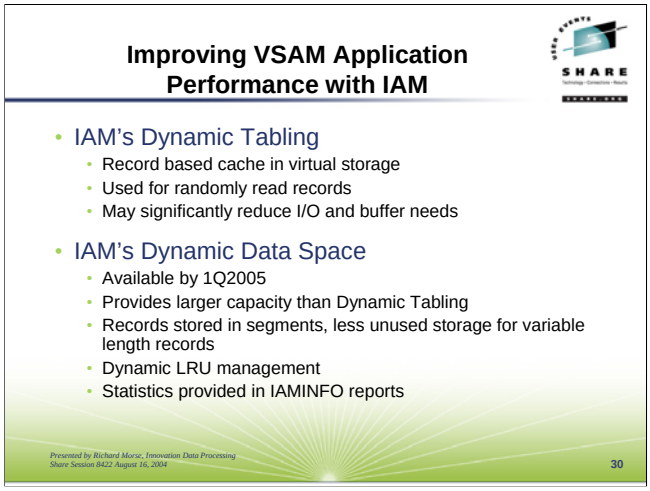
With IAM's Real Time Tuning feature, tuning is simplified and performance of VSAM applications is improved. While results do vary, typical results are in the range of 30% to 80% reduction in elapsed time.



File load buffering is different than file access buffering, because the file load is a sequential output process. IAM defaults to acquiring buffers for two cylinders worth of blocks. During processing, when half of the buffers are filled, IAM issues an EXCP to write out that set of buffers. Concurrent with that I/O, IAM allows the application to continue processing sending records to the other half of the buffers. With the default number of buffers, IAM is writing out one cylinder per I/O during the file load process. Typically a data space is used to hold the index information, however a temporary data set can be used as an alternative.

For Extended Format datasets, IAM uses BSAM to load the file. IAM sets up the I/O so that multiple blocks can be written per physical I/O, however IAM has no direct control over the number of blocks written per physical I/O, as this is determined dynamically by BSAM. For such datasets, the EXCP count for the file load is equal to the number of blocks written out to the file, as opposed to the actual physical I/O count.

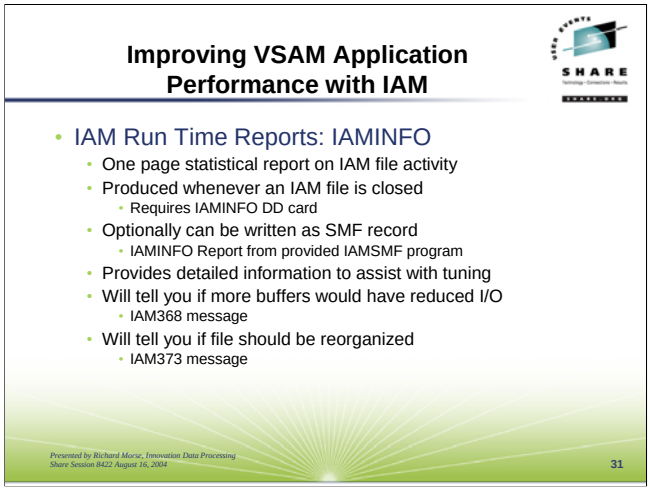
Buffering for IAM file loads is controlled by the CRBUFOPT IAM Global Option or override.



Another performance feature of IAM is the Dynamic Tabling function. This is essentially a record based cache for random I/O activity. When this function is enabled, via an IAM override, IAM sets up a table in virtual storage to hold randomly read records for future reference if the same records are subsequently requested. Once the table is filled, the older records are replaced in the table with the newly referenced records. Updated records are updated both in the table, and within the file itself. This feature has demonstrated significant I/O savings for datasets that have very heavy random read processing.

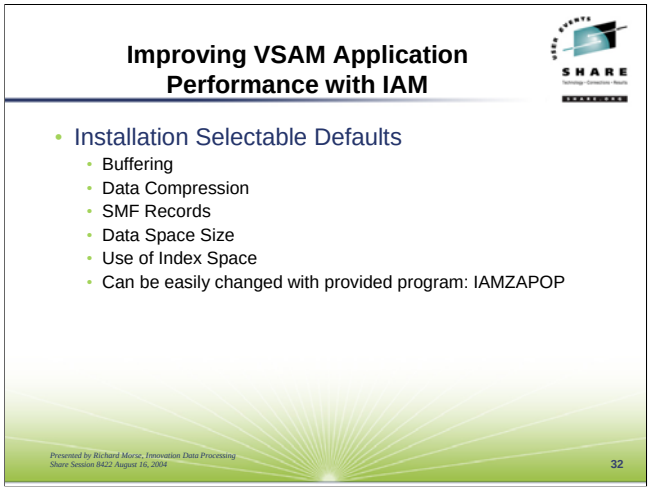
An enhancement for the Dynamic Tabling function has been developed, called the Dynamic Data Space. Rather than using virtual storage from an applications region, IAM will store the data record in a data space. This provides for a larger capacity over dynamic tabling. Additional enhancements include a segmentation of long records so that IAM can use an amount of storage closer to the actual record size, rather than the maximum record size. The records are managed in a dynamic least recently used (LRU) method.

For either function, the user specifies the amount of storage IAM is to use for the table or data space. IAM provides statistical information on the use of this function in the IAMINFO report.

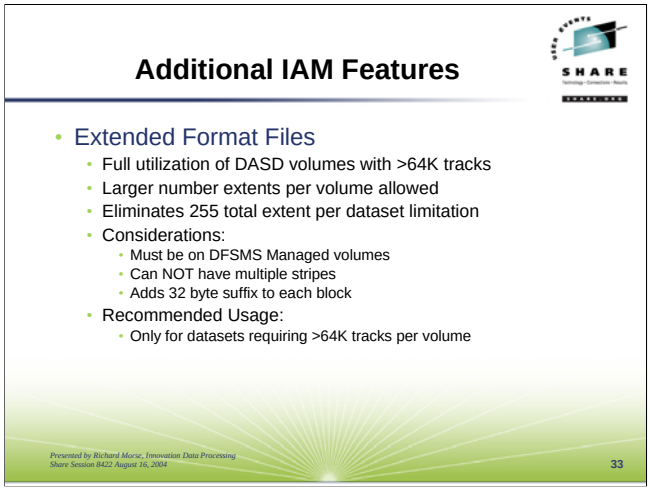


The IAMINFO report provides the detailed information necessary to tune application performance with IAM datasets. The IAMINFO report is a one page statistical report produced whenever an IAM file is closed, if there is an IAMINFO DD card allocated to the job step. Optionally, this same information can be written out to SMF based on the IAM Global Options. If the data is written out as SMF data, IAM provides a program, IAMSMF, to generate the IAMINFO reports from the SMF data.

The IAMINFO report provides detailed information on the file characteristics, resources used for accessing the dataset, and information on the type and quantity of various logical and physical I/O operations that occurred. The IAMINFO report includes information on buffer usage, and will even display the IAM368 message that will tell you if IAM determined that additional I/O savings would be possible by increasing MAXBUFNO. Additionally, the IAMINFO report will produce the IAM373 message if IAM has determined that a file reorganization is recommended to improve performance when accessing the dataset. The IAMSMF reporting program even as an option to only print the IAMINFO reports for those datasets and applications where IAM has detected that more buffers would improve performance.

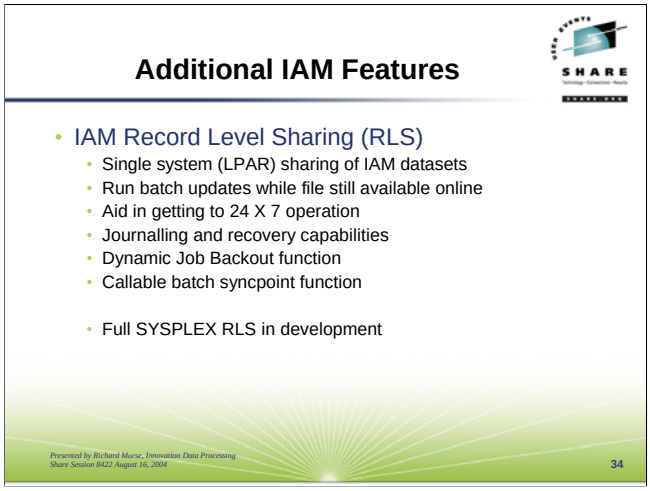


As mentioned earlier, IAM provides the capability for an installation to select various defaults that affect IAM processing. The defaults are kept in what is referred to as the IAM Global Options table. The IAM default options can be easily changed through the use of the provided IAMZAPOP program. There are a variety of options available, which include changing and / or controlling various functions of IAM, in areas such as buffering, data compression, whether to produce the IAM SMF records, the size of the data space IAM can use, and when to use the IAM Index Space.



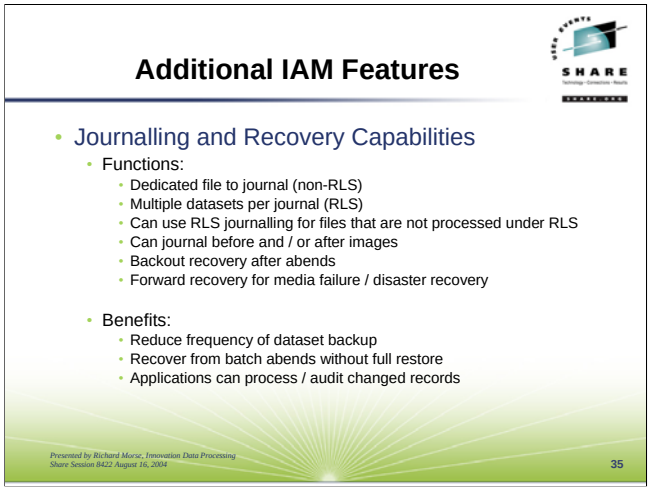
To enable IAM datasets to better utilize large volumes, that is those with more than 64K tracks, IAM provides support for DFSMS Extended Format files. This enables an IAM dataset to utilize more than the 64K tracks per volume, which is the limitation with general non-VSAM datasets. This support also provides support up to 123 extents per volume, and to exceed the 255 total extents per dataset limitations with the general non-VSAM dataset structure. These datasets must reside on DFSMS managed volumes, and can not have multiple stripes.

One thing to be aware of is that extended format datasets have a 32-byte suffix added to each block. IAM will adjust it's block sizes accordingly for extended format datasets to still obtain maximum device utilization, however that means that the IAM block sizes will be 32 bytes less than the traditional non-VSAM dataset block sizes, which can impact the amount of space required for a dataset. For this reason, Extended Format files are only recommended when using IAM datasets on large volumes that will need to use greater than 64K tracks on those volumes, otherwise it is recommended to stick with the general non-VSAM dataset structure.



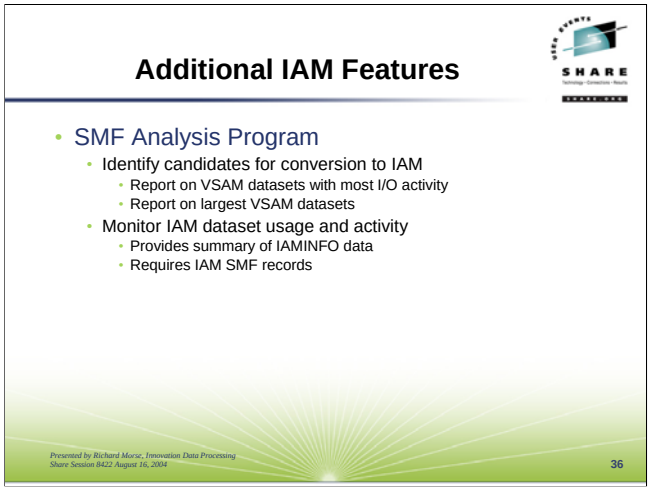
IAM provides for sharing IAM files for update on a single system / LPAR through the use of the IAM Record Level Sharing (RLS) function. This enables customers to run concurrent CICS regions and batch jobs on the same system / LPAR, each of which could be updating the same IAM file. IAM accomplishes this sharing by routing the I/O for the IAM datasets being shared through the IAMRLS address space. The IAMRLS address space provides for record locking without the use of a coupling facility or other SYSPLEX services. Additionally, IAMRLS provides for journalling and recovery capabilities, which includes an available dynamic job backout function should a job step abend. In general, this facility of IAM can be used without changing application programs or JCL. The use of IAMRLS does require the installation of some exits under CICS. Additionally, batch jobs that perform a lot of updates on any recoverable IAM datasets will most likely have to incorporate calls to the IAMRLS batch syncpoint function, to avoid having large quantities of records locked.

Full SYSPLEX record level sharing support is currently in early development stage.

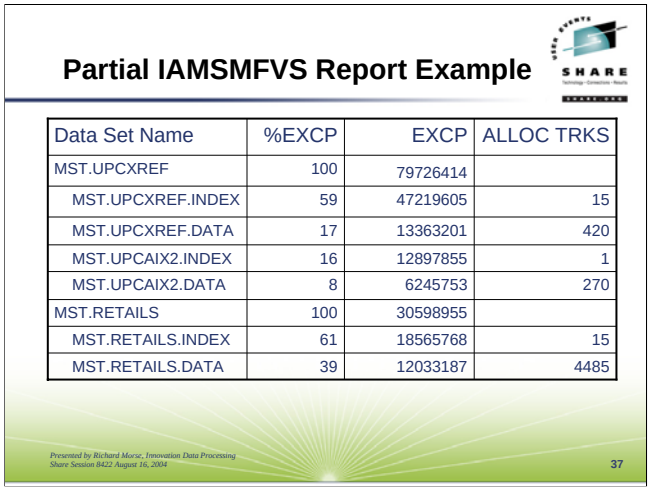


IAM provides journalling and recovery capabilities. This can be performed either using a dedicated journal dataset for each IAM dataset using journalling, which is for applications not using IAMRLS, or multiple IAM datasets per journal under IAMRLS. The IAMRLS journalling is also available for datasets which are not being processed through IAMRLS, however the IAMRLS address space has to be active with journalling active. IAM can journal before images, used for backing out after failures, after images, used for applying updates to a restored copy of a dataset, or both before and after images. IAM provides a journal utility and journal recovery program, or customers can decide to use their own programs to process the journalled data.

Using IAM journalling, an application can reduce the frequency of backing up large multi-volume IAM datasets. This is done for example by backing up the journal dataset that is collecting after images on a daily basis, and backing up the full dataset less frequently. Should the dataset need to be recovered, it can be restored, and then brought up to date with the records from the journal. Another feature is recovery from failed batch runs by using the before images and backing out the updates rather than performing a full data set restore, and possibly rerunning successful processing. Applications can also use the IAM journals for either processing only changes to datasets, and / or use the journals for auditing purposes.



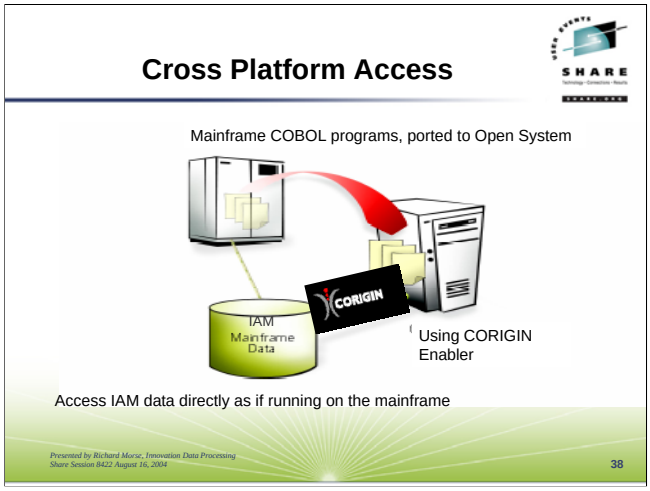
IAM offers an SMF analysis program, that uses SMF step termination records (Type 30 subtype 4 or Type 4), the VSAM records (Type 64), and for IAM dataset reporting, the IAM SMF records. The analysis program produces an TOP 100 EXCP report for VSAM datasets, and for IAM datasets, a TOP 100 SIZE report for VSAM and IAM datasets, and a dataset summary report including all datasets in dataset name sequence. The reports can be used to identify the VSAM datasets that would likely receive the most benefit from IAM in I/O savings and / or space savings. The IAM reports are useful for monitoring IAM usage, and spotting datasets where one might want to check the more detailed IAMINFO reports for tuning, or identifying datasets that might need a reorganization.



This is a partial example from the VSAM EXCP report produced by IAMSMFVS, the IAM SMF analysis program. Just a few of the key columns from the report are shown, which include the data set name, the percentage of EXCP's for each component within the cluster, the actual EXCP count for the component, and the tracks allocated to the component. Below, in the notes, is a sample of the analysis of the report data.

MST.UPCXREF This dataset has an AIX, and is an outstanding candidate for conversion to IAM. While it is a small dataset, it does have very high I/O activity. The combined index component I/O for the base cluster and the alternate index totals 75% of the I/O for this cluster, which will be eliminated by IAM. Additional I/O savings are anticipated for the data components of both the base cluster and the alternate index. The base cluster had some insert activity of just over 3% of the total records with some CI and CA split activity, providing further potential advantages that can be realized by conversion to IAM. The insert rate on the alternate index was lower, however the updates were higher indicating that this alternate index has non-unique keys, again easily handled by IAM.

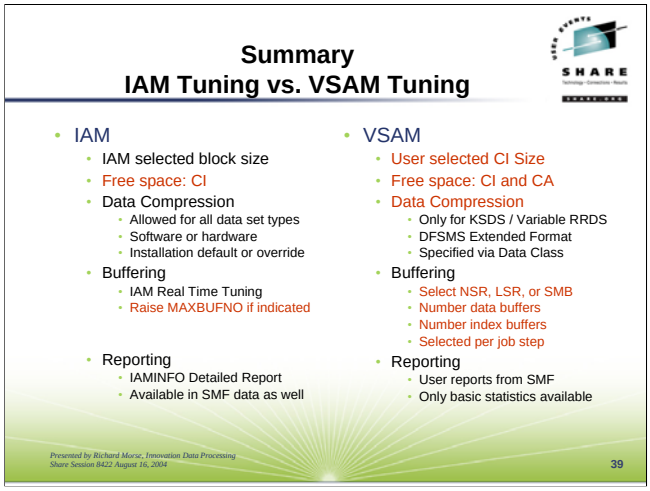
MST.RETAILS Another good candidate for conversion to IAM, with an immediate savings the index component I/O, which is 61% of the total I/O for this file. Additional I/O savings are anticipated for the data component. The insert rate was less than 1% of the total records, however this dataset did experience some CI and CA split activity as well, providing further potential savings from conversion to IAM.



IAM data is universally available…

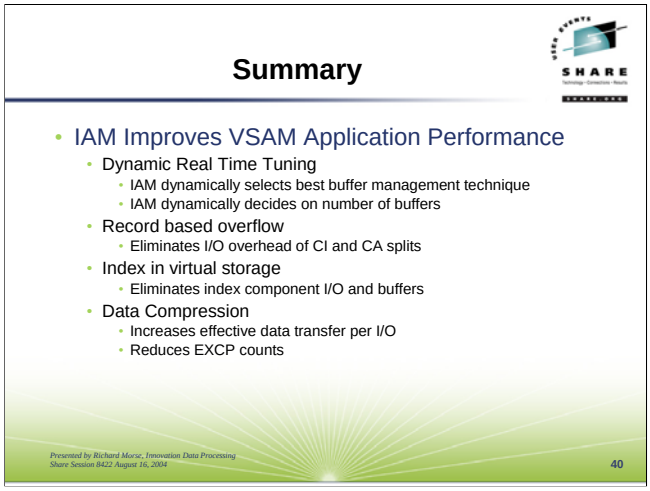
Developed in partnership with Innovation Data Processing, Corigin, a global provider of cross platform data access solutions has “Enabler” software available today for directly accessing IAM mainframe data.

CORIGIN Zero Overhead Access™ technology, enables any program written in a mainframe language that can be compiled on an open system, such as COBOL, FOCUS, and others, to be easily re-hosted to UNIX or Windows platforms without the headaches of having to also move the data that those application access.



It is easier to get the best performance from IAM than it is from VSAM, as shown in the above comparison. For IAM, the main tuning involves raising the maximum number of buffers IAM can use, and for some datasets modifying the CI free space. It is rare when one finds need to change the block size of an IAM dataset for tuning purposes.

With VSAM, there are a larger number of sensitivities, particularly with buffering. In addition to selecting a buffering technique, there is largely manual tuning effort of adjusting number and type of buffers. In addition to CI free space, there is the CA free space to be considered, along with selecting a good CI size for the data component.



IAM has a number of functions that can result in improvements in VSAM application performance, without modifying application programs. These improvements include reductions in physical I/O, potential CPU time savings, DASD space savings resulting in reductions in batch job elapsed times and reductions in transaction response times. For many files, these performance advantages are achieved just by converting a dataset to IAM. Additional savings may be obtained through some simple tuning, guided by the IAM run time reporting.

The key IAM functions that help to achieve these savings include the IAM Real Time Tuning, the record based overflow structure, keeping the index in virtual storage, and the IAM data compression function. IAM has a number of other features that many users find beneficial, in addition to the performance enhancements, such as journalling, single system record level sharing, and access to data from other platforms.


SHARE
Improving VSAM Application Performance with IAM
The End
Session 8422
SHARE
USERBLUE