Paper #4005

MPE/iX Disk Storage: Looking Under The Hood

By

Jerry Fochtman
Bradmark Technologies, Inc.
4265 San Felipe, Suite 800
Houston, Texas 77028
713/621-2808
Jfochtma@mail.bradmark.com
www.bradmark.com

and

Paul Wang
Solution Soft Systems, Inc.
2350 Mission College Blvd, Suite 715
Santa Clara, CA 95040
408/988-7378
PaulWang@solution-soft.com
www.solution-soft.com

Abstract

Proper Disk Storage Management is an area of computer center management that most often is overlooked. There is more involved in properly managing data storage than in maintaining an adequate amount of free space. Proper management of the computer's primary storage medium is a significant factor in influencing the overall performance of the system.

This presentation will help attendees better understand the MPE/iX Disk Storage Management functions and the affect it has on overall system performance. Attendees will receive information covering:

- MPE/iX Extent Management Rules,
- Improving Application Resiliency,
- Efficient Use of Disk Space,
- Application/System Performance Considerations

Armed with this information, user are better positioned to make more effective choices in management of the data storage resources.

Terminology

Before examining how MPE/iX manages disk space, it would be useful to do a quick review of the various terms that are used to describe aspects of file management.

Page: A unit of disk storage comprised of 4096 bytes, or 16 sectors.

Disc Extent: A chunk of contiguous disk storage space. It is allocated in units of pages. A disk file is composed of zero, one or more extents.

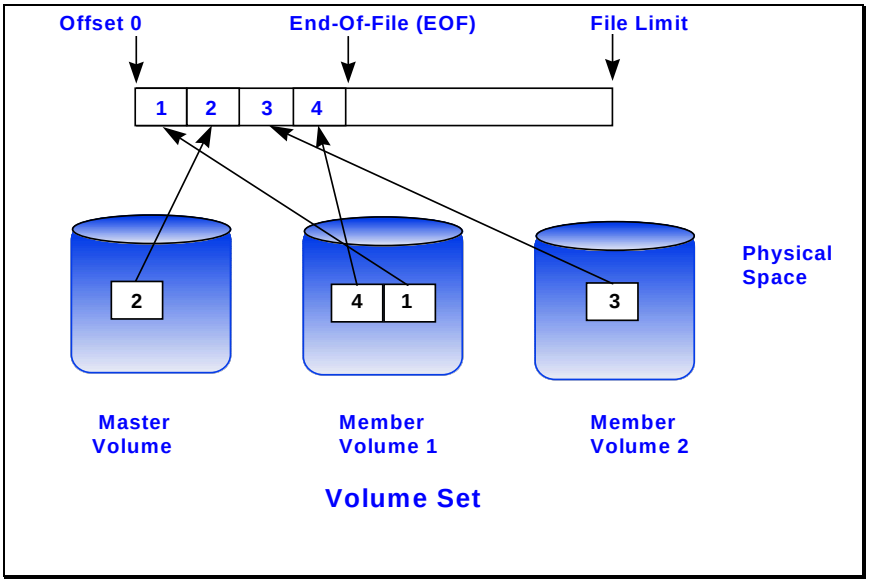
Disc Volume: This is an individual disc drive. Every volume is required to have an assigned name. Also, a volume that is in-use can only be a member of a one volume set.

Volume Set: A group of one or more individual disc volumes. A volume set consists of a *master volume* and possibly one or more additional disc volumes termed volume set members. The MPE/iX operating system requires that one system volume set with the name of 'MPEXL_SYSTEM_VOLUME_SET' exist. Optionally, up to 32 non-system, or user volume sets can be added to a system.

Volume Class: This is a name assigned to a subgroup of individual volumes in a volume set. It is possible to assign an individual disc volume to more than one subgroup within the volume set by assigning it multiple volume class names.

Directory: This is a set of data structures which keeps track of the location of files. There is a *Master Directory* on the System Volume Set and individual volume set directories on user volume sets. The location of the group and account of a file defines the volume set where the file is stored.

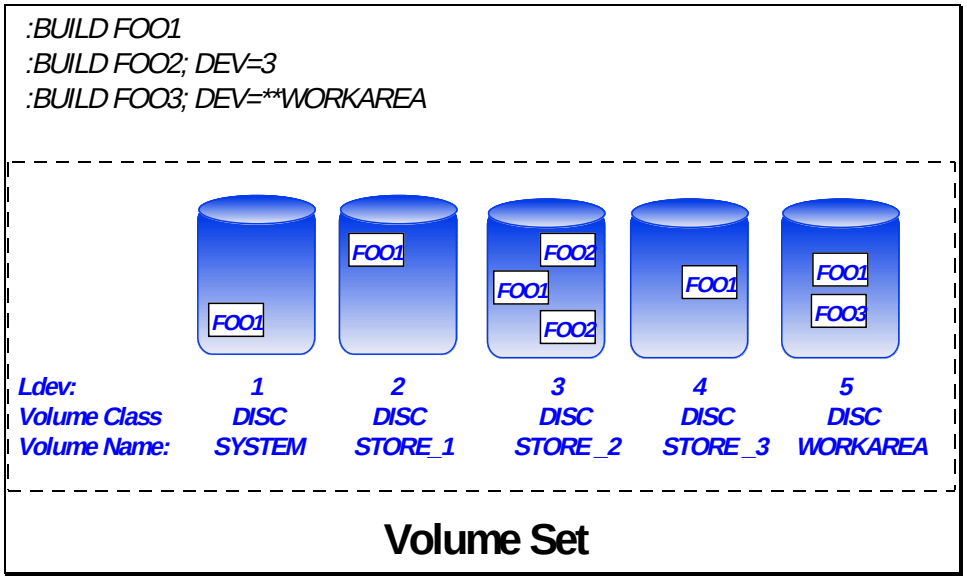
The following illustration shows the relationship between these various components of disk storage:



File Extents

As indicated earlier, files are composed of zero, one or more disk extents. MPE/iX volume storage management also imposes several restrictions involving the placement and size of a file's extents. First, all extents of a file must reside on the same volume set. It is not possible to have the extents of a file span multiple volume sets. However, it is possible to restrict a file to a subset of volumes on the particular volume set.

All files have an attribute called an ***Extent Placement Restriction***. By default, all files are created using the volume class "DISC" as its placement restriction. By specifying a volume class other than "DISC" during initial creation, a file's extents can be restricted to a subset of the volumes within a particular volume set. As such, a user-specified extent placement restriction must identify a subset of volumes within the volume set where the file's group and account are defined.



Determining Extent Size

There are four basic situations where the size of an extent is determined. First, during file creation an initial allocation size may be specified. This causes the file system to allocate one or more extents to satisfy the initial allocation request. Second, the STORE/XL utility and MPE/iX COPY command use an allocation method called ***contiguous block***. This high-speed technique for transferring data determines the size of a file's extents.

Next, the new Dynamic Dataset Expansion feature of IMAGE/SQL will expand the size of the dataset attempting to use a single extent to satisfy the increment value. If not possible, multiple extents will be used until the requested expansion is satisfied or available disk space is exhausted. Finally, there also is the situation where an access request is made for a portion of a file which has not been previously allocated. In this case an extent fault occurs and a new extent must be dynamically allocated before the access is allowed to proceed.

Initial File Allocation

At file creation, it is possible to specify the amount of space to initially allocate to the file. This is normally achieved through the use of the BUILD command, HPFOPEN and FOPEN intrinsics, and also when DBUTIL builds an IMAGE/SQL database from a compiled schema, or rootfile.

Using the BUILD command, the user simply supplies the third sub-parameter to the 'DISC=' keyword. This parameter indicates the number of extents to be initially allocated. The size of the extent is determined using a formula that is based upon the computed total size of the file ((record width x file limit) + (max user labels * 256)).

The HPFOPEN and FOPEN intrinsics also provide an option for specifying the number of extents to initially allocate when creating a new file. With the FOPEN intrinsic the user indicates the number of extents that are to be allocated. However, the HPFOPEN intrinsic allows the user to specify the amount of space to initially allocate based upon a record count (option 36).

Finally, when creating an IMAGE/SQL database, DBUTIL forces the complete allocation of all the space needed for the each file comprising each dataset in the database. This is done based upon IMAGE's expectation that the physical storage space for the dataset capacity is available.

STORE/XL and MPE's COPY Command

STORE/XL and MPE's COPY command use an technique called 'contiguous block'. This involves obtaining a virtual address range which is associated with disk space. Each of these ranges is declared to be a contiguous block. The actual physical disk storage area associated with the block is not necessarily a single contiguous extent, but may be composed of several separate extents. A list of these contiguous blocks is developed which represents the data area contained in a file. For example, a sparse file contains five - 4Kbyte extents. The first four extents hold the first 16 Kbytes of data in the file. The fifth extent is at the end of file and the file's EOF contains 40 Kbytes. There is no data in the file between the forth extent and the last extent with the EOF. In this case, the file's contiguous block list contains 2 entries; one for 0 to 15 Kbytes, and the other from 36 to 39 Kbytes. As such, the contiguous block list represents logically contiguous portions of a file.

During a copy or restore, disk space will be allocated according to the contiguous block list. So space will be allocated for the first 16 Kbytes and then again for the last 4 Kbytes. But no space will be allocated or copied for the intervening 20 Kbytes. In this manner, non-existent data portions of a file, or the gaps between extents in 'sparse' files, are not copied.

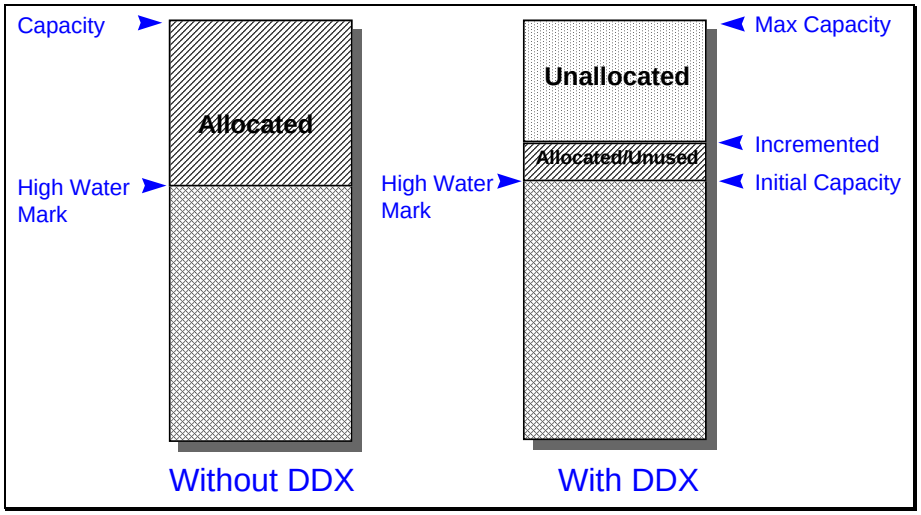
When posting the virtual space to disk, the file system attempts to allocate a single extent to each contiguous block in the list. If unable to locate a large enough chunk of contiguous free space, the largest available chunk of space is allocated for the first extent associated with the virtual address range and the search proceeds for a chunk of free space to hold the remainder. This continues until each individual contiguous block in the list is posted to disk.

Because of this technique, not only are unallocated areas within a file ignored, files with multiple extents can be restored/copied to new files with larger extents. If sufficient large chunks of contiguous free space are available, a logically contiguous file could be placed in a single extent.

IMAGE's Dynamic Dataset Expansion

Introduced on MPE/iX 5.0, Dynamic Dataset Expansion (DDX) was a technique which allowed IMAGE/SQL detail datasets to expand and allocate more disk space for the file as the volume of data in the file grows. When setting a dataset for DDX, the user specifies an initial dataset size, the maximum size allowed and an increment to use when expanding the set as it grows from initial size to maximum size. Using this IMAGE/SQL feature, users do not have to incur the cost of consuming large amounts of disk storage for data which is not yet present in the dataset.

When initially built, DBUTIL only allocates space for a DDX dataset that is needed to contain the initial capacity. Then, as more data is added to the set and its active entry count equals its current capacity, IMAGE/SQL expands the capacity of the file by the increment value in order to add the next entry. These periodic expansions continue until the active entry count reaches the maximum capacity or the expansion process fails because there is insufficient disk space to accommodate the expansion.



Dynamic Extent Allocation (Extent Fault)

Perhaps the most frequent method used to allocate disk storage is through dynamic extent allocation. Whenever access is attempted to a portion of a file which has not previously been allocated, the process encounters what is defined as an ***extent fault*** and an extent for that area of the file must be allocated before the access is allowed to proceed. This most often occurs when a process is adding data to a file beyond its current EOF.

The size of the extent to be allocated is determined by the total maximum space of the file (file limit x record size). If this value is less than 64 Kbytes, the extent size is set to this value. In this manner, small files are generally allocated in a single extent. If the total file space is between 64 Kbytes and 2 Mbytes, the extent size defaults to 64 Kbytes.

Finally, when the total file space exceeds 2Mbytes, the extent size is calculated using a formula. The formula is such that as the file grows, the sizes of the extents grow until a specific maximum size based upon that file's characteristics is reached. At that point, any additional data placed in the file results in the maximum extent size of that file being allocated. The formula used is:

Extent size = MIN[MIN(512KB, *1/32 of file size*),MAX(64KB, *Current Allocated Space*)]

To illustrate how this formula works, the following example shows the extent sizes for a file whose total space requirements are 32 Mbytes:

Min [Min (512K, 1204K), Max (64K, 0K)] = 64K
64K = 64K
128K = 128K
256K = 256K
512K = 512K
1024K = 512K

always 512K from now on.

In this example, a smaller, 6.4 Mbyte file is being allocated. Notice that the maximum extent size for this file is different:

```
Min [ Min (512K, 204K), Max (64K,  0K) ] =   64K
                                  64K    =   64K
                                 128K    =  128K
                                 256K    =  204K
                                 460K    =  204K
                          always 204K from now on.
```

An upper boundary of 512 Kbytes was established to try and reduce wasted disk space when only a portion of the last extent contained data.

Contiguous Block vs. Multiple Extent Allocation

There are a number of differences between the contiguous block technique used by STORE/XL and COPY as compared to the multiple extent allocation approach. The most obvious difference noticed by users is the difference in performance. The first reaction most users have is why not always use the contiguous block technique and allocate large extents for a file. Unfortunately, large extents are not always that efficient when it comes to reading the file, especially using random access in an intensive, multi-user environment.

Placement of File Extents

There are two basic methods which are used to determine where an extent is placed on a volume set. These are identified as 'non-spreading' and 'spreading' sizing algorithms. The requested size of the extent to be allocated determines which method the file system will utilize.

Non-Spreading Extent Placement Algorithm

When the requested extent size is less than 16 MB, the file system utilizes the non-spreading algorithm for determining extent placement and attempts to allocate the requested size in a single extent. There are 5 basic steps involved in the algorithm:

- Develop a list of candidate volume(s) from those in the home volume set based upon the file's extent placement restriction.
- Sort the list in descending order based upon the ratio of free space to capacity of each individual volume.
- On MPE/iX 5.0, if LDEV 1 is in the list of candidate volume(s), place it at the end of the list. For MPE/iX 5.5 and system volume sets with more than four volumes, LDEV 1 is always placed at the end of the candidate list.
- Choose the first volume in the list which satisfies the requested amount of space.
- If the requested amount cannot be satisfied with a single volume, select the first volume on the list and allocate the largest contiguous chunk of free space for the request.

The following examples show which volume is chosen to satisfy an extent request using a volume set composed of 3 volumes:

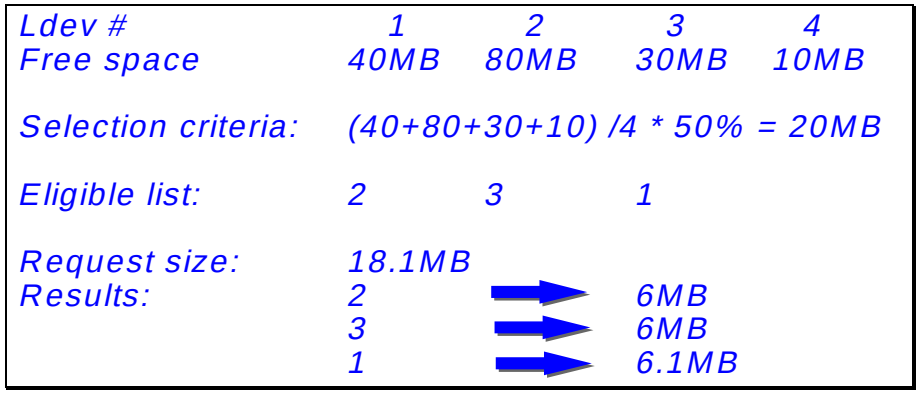
Ldev#	3	2	1
Biggest Free Block	2 MB	4MB	6MB
Free Space Ratio	20%	15%	40%
1 MB Request		Ldev 3	
3 MB Request		Ldev 2	
5 MB Request		Ldev 1	
7 MB Request		Ldev 3	

Spreading Extent Placement Algorithm

When the requested extent size is 16 MB or greater the request is spread into multiple extents using a second algorithm. In this technique additional criteria is applied in selecting candidate volumes as well as the size of each extent created to satisfy the space request. The steps are:

- Develop a list of candidate volume(s) from those in the home volume set based upon the file's extent placement restriction.
- Reduce the candidate list to only those volumes which have at least 50% of the average free space of all eligible volumes on the list.
- Sort the list in descending order based upon their amount of free space.
- On MPE/iX 5.0, if LDEV 1 is in the list of candidate volume(s), place it at the end of the list. For MPE/iX 5.5 and system volume sets with more than four volumes, LDEV 1 is always placed at the end of the candidate list.
- Calculate an extent 'spread size' for each volume by dividing the size of the request by the number of eligible volumes which contain > 1MB of free space. There is an overriding minimum size of 256KB.
- Cycle through the eligible list of disks using spread size as the extent size as long as the remaining needed size >= 1.125 * spread size. Once the remaining size drops below 1.125 * spread size, use the remaining size as the size for the last extent to allocate.

The following example illustrates how the spreading algorithm functions with a request for 18.1 MB:



Allocation Initialization Considerations

When an extent is initially allocated, it is not always initialized. There are some basic rules that are followed. In the case that an extent is being allocated within the EOF of a file it indeed must be initialized with the appropriate fill character. When an extent is being allocated beyond the EOF it is not initialized. However, when the EOF of a standard file is extended, all intervening allocated file extents between the old EOF and the new EOF are initialized.

This initialization process is performed using blocked I/O. As such, this can be a fairly expensive operation for large file extents. This is why extending a file's EOF and then writing data into the new area requires twice as much I/O when compared to simply writing serially beyond the EOF and then updating the EOF when done.

The initialization character used for these fill operations is an attribute of the file. It is established when the file is created. The default initialization character that is used for ASCII files is a space, and for binary files, the a binary zero character is used. However, it is possible to specify a different character value using the HPFOPEN intrinsic when creating a file.

Improving Application Resiliency

Disk failures are going to occur, its only a matter of timing. Technology continues to try and reduce the potential of a failure and also the impact when a failure occurs. However, the possibility still exists, and along with it so do the opportunity to lose data.

Hardware redundancy as well as various software approaches are being used in an attempt to minimize any impact the eventual failure will have on a user's business operation. Obviously large, 24x7 operations spend a great deal of effort and resources to eliminate as much of the risk of data loss. Smaller operations which don't require 24-hour operation still have the same concerns, only less resources to devote towards reducing the risk of data loss.

The most common method used to help reduce the risk of data loss is periodic backups. Although there are a number of issues surrounding tape backup, such as media reliability, validation, etc., this is what a majority of sites rely on for recovery from a disk hardware failure. They are still exposured to a potential of data loss should a hardware failure occur between backups.

The good news, however, is that there are some fairly simple things that can be easily implemented to help reduce this exposure, even when relying on tape backup. These same techniques will help the more sophisticated operations as well.

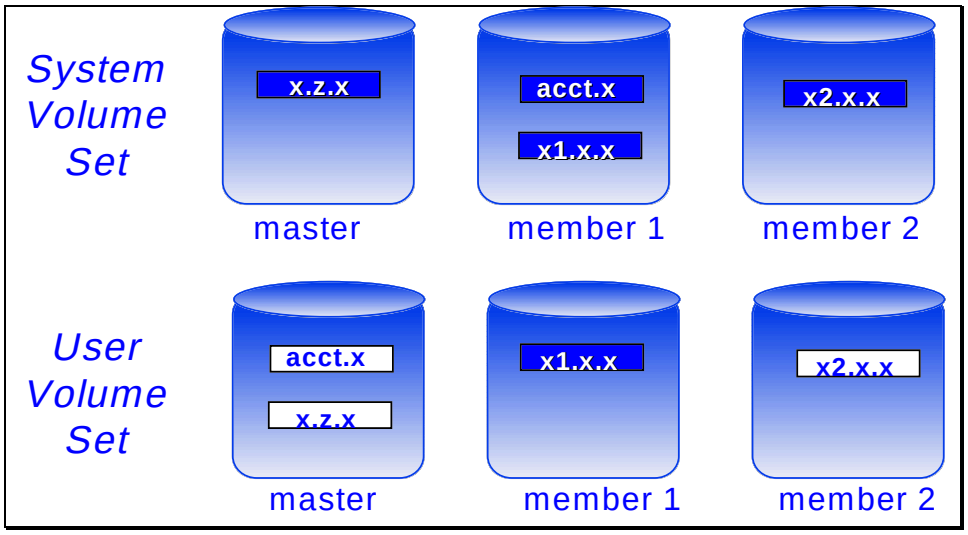
Application Resiliency Tip #1

User volume sets are move forgiving if an individual volume should fail. If the master volume of any volume set, either the system volume set or a user volume set should experience a hardware failure, all files on that volume set are lost. However, there is the possibility that the volume that fails is not the master volume.

If the member volume that fails is a part of the system volume set, all files on the system volume set are lost. This is because MPE/iX requires that all volumes on the system volume set be on-line and available in order for the volume set itself to be available. Any files that resided on other, unaffected volumes on the system volume set cannot be recovered.

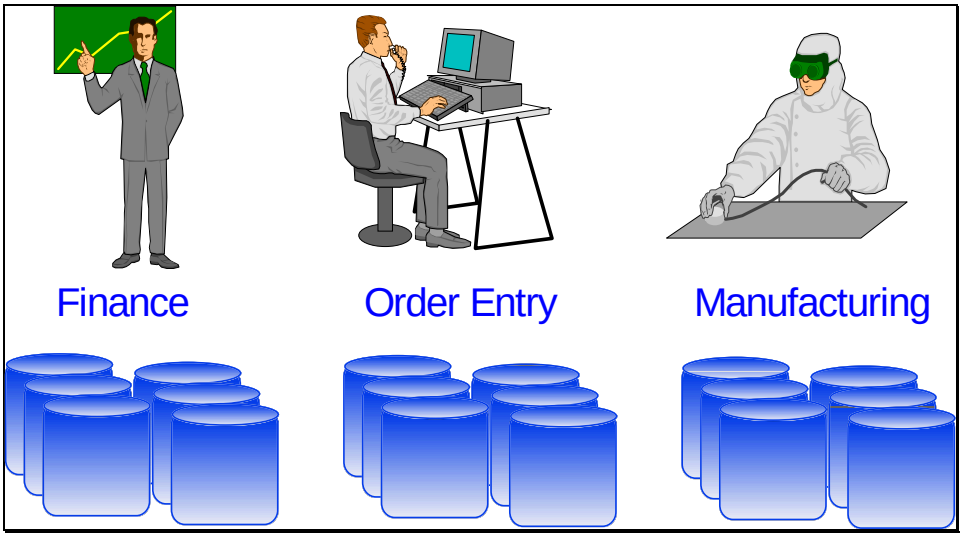
User volume sets are slightly more gracious when it comes to the loss of a member volume. Only those files which had extents on the failed volume are not recoverable. All other files which did not have extents on the failed volume set member can be accessed. As such, one can get a backup of the unaffected files, or simply replace the defective drive and restore the files that had extents on that drive.

To illustrate, the following examples shows what files are still accessible given the loss of *member1* from the volume set. In both cases, file *x1.xx.xx* cannot be accessed. However, in the case that the lost volume member was a part of the system volume set, all other files cannot be accessed as well, which is not the case when the lost volume is a part of a user volume set:



Application Resiliency Tip #2

Separate critical applications on different user volume sets. Given that a user volume sets provide a better degree of recoverability, placing major, critical applications on separate user volume sets will further minimize the impact a volume failure will have on a business. In this arrangement, should a member from the volume set for one application fail, the other applications running on separate user volume sets will be unaffected and can continue to operate.



Using separate user volumes can also be useful when consolidating computers. Simply use one or more user volume sets for the applications and data from each separate system. If need be, use the extent placement restriction to separate applications within the same user volume set.

Application Resiliency Tip #3

Separate IMAGE logfiles from their corresponding database onto different volume sets. Many users employ database logging so it can be used as a vehicle to recover a database that is damaged between backups. However, many DBA's unknowingly subject the logfiles that they will rely upon for recovery to the same potential failure that will impact the database they are trying to protect. To ensure every opportunity for recovery, the logfiles themselves must be separated from the same physical disk drive(s) which contain the source database.

It is possible to use extent placement restriction and build the first logfile for the logset on a different volume class of the same volume set where the database resides. However, because the extent placement restriction is a file attribute, the logging facility does not know anything about it when automatically creating subsequent logfiles. As such, automatically created logfiles may be on the same volume(s) as the source database. Should a volume failure now occur, not only could the database be lost, but also possibly one or more of the logfiles needed for the roll-forward recovery as well.

To ensure recoverability using roll-forward recovery, the logfiles themselves should not be on the same volume set as the database. To achieve this, build the logfiles on a separate

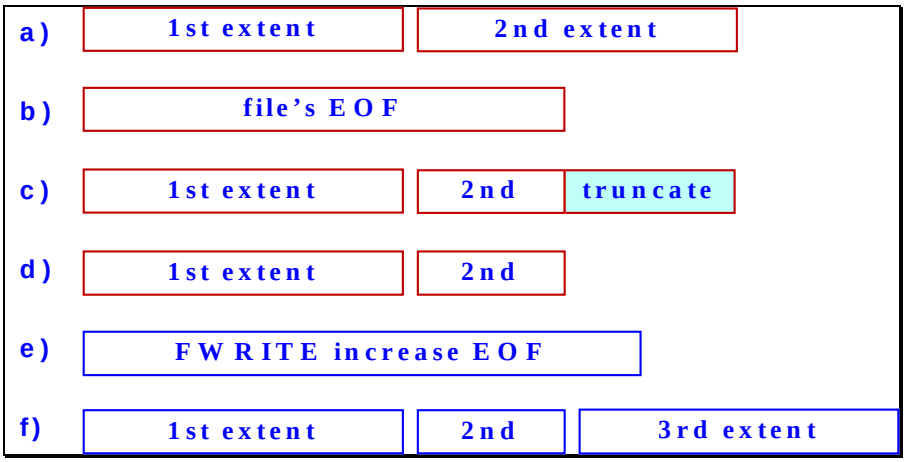
group and/or account located on a different user volume set. In this manner, if the volume set hosting the database should fail, the logfiles on the separate volume set could be applied to the last backup to recover the database. If the volume set hosting the logfiles should fail, the source base could be backed-up and logging restarted to a separate user volume set.

Efficient Use Of Disk Space

There are a number of different steps one can take to improve the use of existing disk storage space. Many of these steps can be done by making simply adjustments in how files are created and managed. However, some options are better addressed through the use of an available third-party tools.

Disk Space Efficiency Tip #1

Truncate files periodically. There is a fairly large amount of wasted space beyond a file's EOF. This waste can amount upwards to 512 Kbytes, depending upon the extent allocation size of the particular file. As such, periodic truncation of all files will free-up this wasted space, returning it back to the free space pool. Subsequent additions to the file, expanding the EOF will cause the file system to allocate another extent to the file to contain the added data. As such, file truncation increases overall disk fragmentation.



Disk Space Efficiency Tip #2

Utilize Dynamic Dataset Expansion for IMAGE datasets. IMAGE datasets fully allocate all space up to the user-specified capacity. As such, any space in excess to what is needed to contain the actual entries is unused and wasted. Because of the general rules of keeping detail set free space at 15-35% of capacity, this could amount to a fairly large amount of wasted space for large datasets.

To reduce the amount of space wasted, consider enabling Dynamic Dataset Expansion for the IMAGE detail datasets. As outlined earlier, the dataset will automatically expand when there is no more available space to add additional entries (see *IMAGE's Dynamic Dataset Expansion* on page 4).

Disk Space Efficiency Tip #3

Purge or archive files that are no longer needed. Taking an active role in managing the files on a system is an easy way to mange the amount of available free space available on a system. Too often files containing extracted data for a report, program test data, multiple versions of source or data files, left over output reports, etc., are not removed from the system when the task is complete. Instead we tend to move on to work the next problem, with good intentions of cleaning-up when we have a moment. Before we realize it several months have elapsed and we no longer remember what the particular files were for, or the person is no longer around and we're leery of purging them for fear that they are needed for an application.

Obviously nothing beats a good change management process to ensure when the task is complete everything is appropriately removed or placed in the proper group/account. A program of periodic archival of data and then reducing the capacity of datasets will not only help with providing free space, it may also help improve application performance as well as reducing backup time.

Disk Space Efficiency Tip #4

Do not inflate file limits when creating files. When the actual amount of space needed for a file is unknown, most users specify a fairly large FLIMIT for the file so as to have plenty of available space. However, as outlined earlier, a file's FLIMIT plays an important role in determining the size of and method for allocating extents to the file. When an inflated FLIMIT is used it is more than likely that much of the space in the final extent will be wasted.

Disk Space Efficiency Tip #5

Do not over allocate space when initially allocating a file. Another opportunity for efficiency comes in when specifying an initial allocation for a file. It is quite easy to force the allocation for more space than is actually needed. Furthermore, during creation all space for the requested allocation will be initialized, adding additional I/O overhead at creation. As such, consider not forcing any initial allocation and allow the extent fault management to expand the file as warranted.

Disk Space Efficiency Tip #6

Use the right disc drive as LDEV 1. Finally, be selective when assigning the type of disk drive to be used as LDEV 1. There is a hardware constraint in terms of the amount of addressable space used on LDEV 1 depending upon the I/O channel type on your system. For the older, CIO channel-based systems, the largest addressable space for LDEV 1 is 2GB. For the newer, NIO channel-based systems, it is 4GB. So if you assign a 4GB drive as LDEV 1 on CIO-based systems 2GB of available space would not be used. If one used the upcoming 9GB drive for LDEV 1 on an NIO-based system, almost 5GB of storage space would be wasted. So be sure and follow HP's guidelines in terms of using the proper disk drive for LDEV 1 on your system.

System Performance

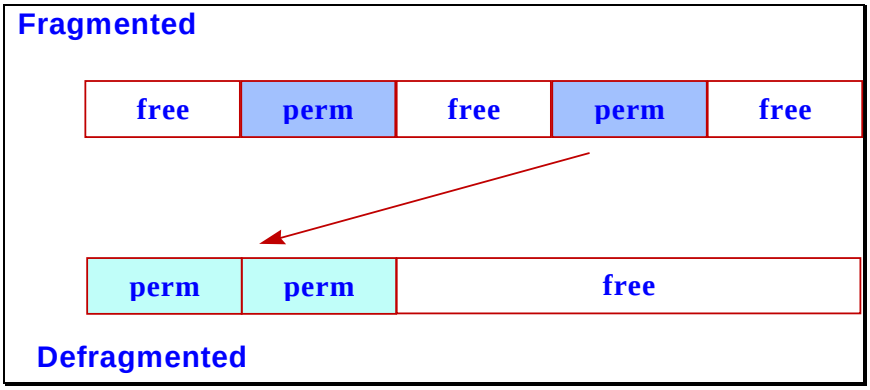
Along with increased efficiency, there are a number of steps that one can perform related to disk storage management which will help in the overall performance of the system. To perform the tasks necessary to address these issues one has to either perform a reload their target volume set or employ one of the available 3rd party tools. The following performance tips apply regardless of which option one might choose:

System Performance Tip #1

Defragment the disc volume(s) periodically. Because of the dynamics of allocating and deallocating extents, it is quite easy for a disk system to become more and more fragmented, resulting in smaller areas of free space spread through-out the disk volume. As this happens, and more files are created, there is an increase in the amount of I/O needed by a system to access the growing set of new or expanded files. Also, as the sizes of free space continue to shrink, once they are below 64KB in size, they will more than likely be wasted as they are no longer considered for allocation for file sizes > 64KB. When this occurs, even though one may have a lot amount of available free space in these small 'chunks', it is common to see 'out of disk space' errors when attempting to build larger files or invoke utilities such as HPSORT.

To reduce I/O rates and encourage larger, more efficient I/O, disk storage should be periodically defragmented**.** Not only does this eliminate the possibility of a premature 'out-of-space' error, but also provides utilities such as HPSORT with large contiguous areas to work with, thereby enhancing its performance. In addition, operating system updates require a large amount of free space to properly update the system. Defragmenting the system volume set is one way to help achieve this requirement.

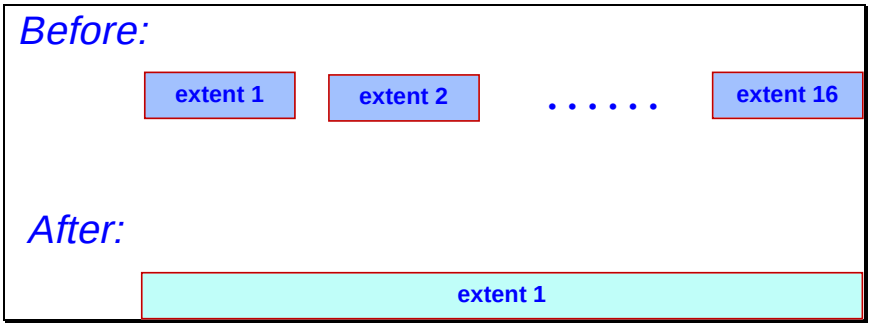
Defragmenting disk storage involves moving existing file extents so as to create larger contiguous areas of free space as shown in this illustration:



System Performance Tip #2

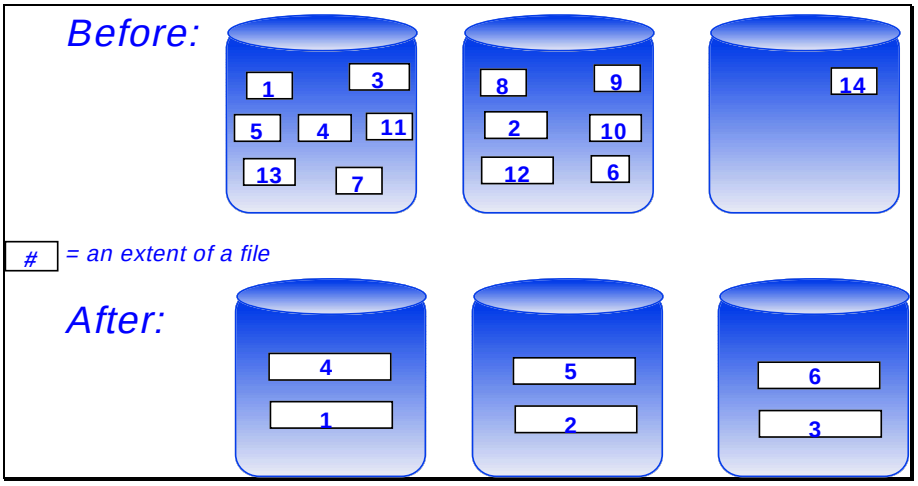
Combine extents for small or predominately serial-accessed files to one or only a few extents. The more extents a file has, the more I/O activity necessary to access the complete file. For large, random access-type files, such as large database files this is desirable in order to maximize overall throughput with multiple accessors. However, for smaller files, or files with predominant sequential access this can actually slow down overall performance.

Files such as program files, UDCs and MPE command files, and those files which are serially accessed it is better to minimize the number of extents they have in order to encourage larger, more efficient I/O performance. Using STORE/XL, COPY or a third-party tool, one should combine the many extents of these types of files into a single extent for overall performance efficiency:



System Performance Tip #3

Spread extents for large, random accessed files across volumes. For large files, almost the opposite is true. Multiple extents help in the overall performance of large files used primarily for random access patterns by a large number of concurrent users. However, distributing the extents across multiple volumes in the volume set encourages more concurrent I/O access. The most obvious candidate files for this technique are IMAGE/SQL, ALLBASE/SQL and KSAM files. Along with spreading the extents across the volume set, combining the extents that are on a single volume set help provide more efficient I/O performance for the individual volume:



If the volume set has more than 5 volumes, consider excluding the master volume from the pool of volumes used in spreading the file's extents. This is achieved by assigning a volume class to all volumes except the master set and then assigning this volume class name as the volume restriction for the file. This is because the master volume contains a number of items related to the volume set, including the directory of the volume set. I/O requests for these items as well as those requests for extents of the target file will compete in the same I/O queue. So separating these activities on very I/O intense files/volume sets will provide an opportunity for overall shorter I/O queues and better throughput.

System Performance Tip #4

Spread files across all discs when adding new volumes to a volume set. New disks are added to a system to increase the amount of available on-line storage. However, this may also result in slower I/O performance on a system. If you recall, one of the attributes used in ordering the available disks when allocating extents is its ratio of free space to its size. For a new disk, this ratio could be close to 100%, making it the primary candidate whenever any file allocates a new extent. This results in all new files or extents being placed on the new disk. Therefore all access to these newly created areas will compete for the same, growing I/O queue resulting in slower I/O performance.

As such, when adding a new disk to any volume set, the files on that volume set should be spread across all the disks, thereby reducing the I/O queue length for any individual disk, balancing the I/O across all disks. This can be achieved by reloading the individual user volume set or by using one of the available third-party tools.

System Performance Tip #5

Don't overload the capacity of the I/O channel. Although an I/O channel can have up to 15, 7 or 8 devices for Fast-Wide SCSI, single-ended SCSI or Fiber Link channels respectively, placing this number of disc volumes on a channel could easily cause I/O performance problems. The total I/O demand may easily exceed the capacity of the channel, thereby causing longer waits for I/O to complete. A good rule of thumb is not to exceed 9 discs for Fast-Wide SCSI, 4 discs for single-ended SCSI and 4 discs for Fiber Link channels. If the channel also shares a busy serial device, such as s tape drive, it may be necessary to have still fewer disc volumes.

System Performance Tip #6

Do not mix disks of differing capacities in the same volume set. Again, because the ratio of capacity to free space is used in the allocation algorithm, mixing disks of different capacity will tend to favor allocation to the larger disk causing it to become an I/O bottleneck for the volume set.

System Performance Tip #7

Utilize STORE or the COPY command to copy files. Not only can these approaches be faster than the tried-and-true read-write approach, but the target file will probably have fewer extents than the source file and ***gaps*** that may exist in the source file will not result in allocated space in the target file.

System Performance Tip #8

When creating a new file and writing to it, always write beyond the EOF before extending it by posting the EOF. In this manner, the process does not incur the added overhead of initializing the space and posting it to disk before filling it with actual data.

System Performance Tip #9

Use the initial allocation of a file appropriately, not over allocating what is needed. While this approach to allocating space for a file is more efficient than dynamic allocation, it is also easy to waste space. One needs to balance both aspects when using this feature.

System Performance Tip #10

Monitor LDEV 1 closely. For small systems, with 2-4 disks on the system volume set and no other user volume sets, it is important to monitor LDEV 1 carefully. If free space is low or becomes exhausted, system aborts will occur.

Conclusion

The extent management aspects of MPE/iX's is both fair and flexible. It offers many opportunities to provide high I/O performance. However, it is still generally automatic and designed to provide a 'one size fits all' solution. With knowledge about your specific applications and how they are used coupled with knowledge about how MPE extent management works, it is possible for you to do follow a number of the tips and techniques presented to improve application resiliency, disk space utilization along with overall system performance.

Biography

Jerry Fochtman is Director of HP3000 Product Development for Bradmark Technologies, Inc. He has over 22 years experience on HP3000 systems, having also served as Chairman for SIGIMAGE and also spent 9 years serving as Chairman for the Greater Houston Regional User's Group.

Paul Wang is the President of SolutionSoft Systems, Inc. He is a software developer and consultant, specialized in transaction management, system performance, file system internals, database and on-line transaction processing. Previously, he was an internal architect of transaction management in HP's Core MPE/iX Lab.

~ END ~