Section IV -- Input / Output

File System -- User Interface

Systems typically spend 5 to 40% of their CPU time creating, deleting, opening, closing and searching for files. These activities, will shall be called "file system transactions" are rather CPU intensive. They are performed by the XQP, which in turn, invokes many calls to the VMS lock management facility.

(Slowed down VMS in Version 4.0

If a VAX cluster is active, the CPU demands of these functions increase significantly due to the volume of inter-CPU communication which may be needed and the increase in the number of lock manager calls required.

File Allocation and Extension

The allocation of space, either to a newly created file, or to expand an existing one, is, in general, a costly function, but if space allocation is done by default methods it produces files which perform poorly.

When creating a file, the normal default in user software is to allocate little or no space and then frequently expand the file as it is written. Truncation of unused space may or may not occur upon file closing. This can dramatically slow the software.

* SET FILE/TRUNCATE

Mechanisms exist to declare the initial amount of space to be allocated and to set an amount which will be allocated each time the file must be extended. Options can be set to automatically truncate upon closure. These minimize allocation efforts by getting it all done in one or a small number of steps, and eliminate wasted space.

RMS_EXTENDSIZE
Set to 40

Files exist as one or more pieces on disk -- each piece is called an "extent" or "fragment". If the file is completely contained in one fragment it is said to be contiguous. (This should not be confused with a file which is "marked" contiguous -- such a file not only must be contiguous, it may not be expanded unless the result would be contiguous.)

.Files which exist in many pieces are said to be fragmented. If the fragmentation is severe, the result is poor performance due to two separate reasons:

"Window Misses"
When a file is opened, information about only ACP_WINDOW fragments (7 by default) is made available to the disk driver. If an access is made outside the area covered by these fragments a "window miss" occurs. To get the needed information requires an access to the disk structure database and 10 to 30 milliseconds processing time (780/8200).

Split I/O's
When an I/O is issued that crosses a fragment boundary, two or more physical I/O's must be performed to complete the request. The extra cost is at least 1 millisecond of CPU time and 5 to 60 milliseconds elapsed time.

Files can be allocated in one of 3 ways:

Default
Allocate without regard to contiguity. The result is generally highly fragmented.

Contiguous
Allocate contiguous space, or fail.

Contiguous Best Try
Allocate file in as few fragments as possible. (VMS has a long-standing bug where it will only allocate the three largest fragments on the disk to the file (if needed), then allocates additional space using the default method.)

Contiguous style allocation may take longer (but only for medium, not very small or very large, size files), but produces significant performance gains when the file is used.

The success and speed of the disk space allocation process is materially affected by the availability of contiguous free space on the disk. If there are no large free areas of space on the disk, allocations, especially contiguous style, take longer. Moreover,

given the bug in the contiguous best try algorithm, if a file can't be allocated in 3 or fewer fragments, the result will be a highly fragmented file. If most files are allocated contiguous best try, it will tend to increase the availability of contiguous free space, since as files are deleted, the freed space will be contiguous. However, over time, the contiguity of the free space will break down, absent periodic steps to re-organize the disk.

In general, the more free space there is on a disk, the easier it is to allocate space to files.

However, there is no valid reason why an application under VMS requires completely contiguous files.

WARNING: When doing a DIRECTORY/SIZE the value displayed as the file "size" is not the file size. It is the RMS EOF pointer, which is maintained only by RMS. It may be of any value whatsoever for a legal VMS file. The true file size is displayed by DIRECTORY/SIZE=ALL.

We have it turned off - good.

VMS includes a mechanism called "Highwater Marking". If on, it causes zeroes to be written to all file space when newly allocated to a file. It is on by default. To turn it off for a given volume: SET VOLUME/NOHIGH.

Directories

Directories are sequential lists of file names with the related File IDs. They must be searched top to bottom. VMS may have available a one letter index per directory block, but in general, long directories require long search times. Files should be organized into small sub-directories. VMS does have an efficient way of navigating sub-directory structures, so long as searches do not have to be run through many sub-directories.

* extra programs in a dir. are wasting resources.

User productivity will increase if the user has files well organized. Long directories lead to wasted disk space and wasted user time.

WARNING: Sub-directory structures can go up to 16 levels deep. Do not allow them to go beyond 8 as levels beyond 8 can not be effectively handled with BACKUP and other utilities. To detect:

*check

```
DIRECTORY dev:[*.*.*.*.*.*.*.*.*]*.DIR
```

Files Opens and Closes

Excessive opens and closes are a frequent programming error. Files should be opened when first needed and closed only when no further use will be made of them.

Use of the File System as a Librarian

Check out more uses for the librarian e) use instead of a directory with large no of entries

Many applications create large numbers of small files, each holding a piece of the database being maintained. Reliance is placed on the file system to organize the data. In effect, the file system is used as a librarian.

This leads to huge directories, long searches, frequent allocations and deletes, and uncountable opens and closes, all of which consume vast amounts of resources. On clusters, the problem is magnified.

top. cluster size 1 and look at the initial = 2. — cluster size = 4 is probably better except for small files. in which case 2 is probably better.

The VMS librarian utility does a much more efficient and controllable job. For very intensive applications a librarian procedure specific to the application should be developed.

Disks in a VAX Cluster

The VAX cluster scheme allows a disk to be open to full access to multiple VAXes simultaneously. This is accomplished by having one VAX act as a "traffic cop" for any given disk. When any other VAX wants to perform a file system transaction on that disk, it must communicate with the traffic cop VAX to get permission. The added cost of this is about 15 milliseconds of CPU time (780/8200) per file system transaction per involved VAX.

In a "wide open" VAX cluster (all disks are open to and used uniformly by each cluster member) this additional processing typically results in an overhead consumption by each VAX of 5 to 20% of its CPU time to support just the coordination of the file system transactions.

In addition, whenever a disk is mounted to more than one node, regardless of which CPU performs the file transaction, extra lock manager requests are used over the case where only one CPU has the disk mounted. Typically there are 2 to 5 extra requests using 3 to 8 milliseconds of CPU time (780/8200) per file system transaction.

Implications

Major files should be created with explicit specification of an allocation size appropriate to the application. Any file of significant size or use should be allocated contiguous best try. Most languages provide mechanisms to effect these options. If not, use RMS directly.

Disks must be periodically reorganized to make all files and the free space as contiguous as possible.

Avoid letting disks run over 90% full if files are frequently allocated on them. Not only will allocation steps be faster and the result (when requested) more contiguous, but there will be fewer cases of run failures due to exhausted free space and, therefore, excessive consumption of resources due to re-runs.

If fragmentation is unavoidable, files should be opened with large windows. Critical applications may use cathedral windowing.

Avoid running wide open VAX clusters. Specifically allocate workload across machines so that each disk is used primarily by one VAX and as little as possible by every other VAX. Make sure

the CPU which is the prime user of any disk is the CPU which manages the disk (eg., is the first CPU to mount it).

Avoid long directories.

Avoid unnecessary directory searches. Example -- frequent calls to PURGE.

Turn off volume highwater marking if security doesn't require it. If security does, a better approach is to enforce the use of SET FILE/ERASE and DELETE/ERASE on sensitive files.

Occasionally do SET FILE/TRUNCATE on files to eliminate wasted space, but only to those files for which the I/O routines maintain the EOF pointer by RMS standards.

don't use for ACL >12 long; the command is totally wipe out part of it.

File System - System Manager Control

Disk Cluster Size

The disk is allocated by groups of blocks called "clusters". The VMS default cluster size of 3 is totally inappropriate due to the default RMS I/O transfer sizes which are in multiples of 4. Larger cluster sizes will improve performance, but waste disk space.

ACP Caches

← Don't touch it!!

The file structure data base consists of the various data defining files and the disk, and an outer layer of file organization called directories. This information is stored on the disk and accessed and manipulated by the XQP.

The XQP has a set of caches against this data in an attempt to minimize actual I/O's when parts of the database are re-accessed by users. (These are called ACP caches because, prior to V4, the functions of the XQP were handled by a separate -- and considerably

more efficient -- process called the disk ACP.) AUTOGEN sets initial sizes for these structures. Occasionally an adjustment of the cache sizes can produce performance improvements in the XQP or will free memory for other purposes.

MONITOR reports the hit rate for each cache; in general, if reducing a cache size doesn't lower the hit rate or the absolute increase in the volume of disk I/O's which results is minimal, memory can be saved and/or hard faults reduced. If increasing a cache improves the hit rate, XQP performance has been improved at the expense of memory. If XQP activity is light relative to the caches concerned, the loss of memory may cost more than the value of the gain in XQP performance.

However, note that, with one exception, the caches are stored in paged system memory. What is reported as a cache hit by Monitor may have caused a page fault, and, if the number of accesses to the cache relative to its size is rather small, the fault may have been a hard fault. Thus the saved I/O is illusory -- in fact, if a hard fault did occur, the cost of the hard fault was more than that which the simple I/O by the XQP would have required. Moreover, a large cache, which is accessed infrequently, will tend to push other pages out of the secondary caches, harming the performance of all work in general.

One should be very reluctant to expand the size of these caches as such may actually worsen XQP performance or overall system performance. An expansion is only mandated if the access rate is very high.

In sum, for the vast majority of cases, adjustment to the size of these caches will have little positive affect on system performance. A typical exception is where the number of file system transactions is abnormally large. However, that, in and of itself, is a significant performance problem which should be directly addressed. Adjusting the cache size is but a temporary expedient which may serve to make a very bad situation very slightly less bad.

The caches and parameters which define their size are:

Directory -- ACP_DIRCACHE

A cache where the actual contents of directory files is stored. If users make frequent file accesses and tend to have large directories (and won't shorten them) or work in lots of directories at the same time (and won't better organize their files), an increase may be helpful, but will consume a large amount of memory. If a relatively small amount of directory space tends to be in use simultaneously, or the rate of file lookups is very low, reduce its size. Note: if users tend to work in many directories simultaneously, an increase in the size of this cache will be useless without increasing ACP_DINDXCACHE.

Extents -- ACP_EXTCACHE

This cache is a list of contiguous empty areas on the disks. If there is frequent allocation of contiguous or contiguous best try files, an increase may be helpful. The space consumed is relatively small. If disks tend to be relatively full, ACP_EXTLIMIT should be raised to about 500, which may help reduce fragmentation.

Index Slots -- ACP_FIDCACHE

A cache of the index to the index file. If file allocation and deletion is frequent, increasing this may help. This is a very small cache.

File Headers -- ACP_HDRCACHE

File headers are 512 byte blocks (generally 1 per file, but occasionally more) which define and describe a file. The cache is useful if individual files tend to be referenced repeatedly.

Disk Cluster Map -- ACP_MAPCACHE

This cache holds information on available disk space. If file allocations and deletes are frequent it may help to enlarge it.

Disk Quota -- ACP_QUOCACHE

This cache should be large enough to hold entries for as many UIC's per disk as are active at any given time and will be allocating or deleting files. Entries are 32 bytes long.

Directory Index -- ACP_DINDXCACHE

This parameter controls the number of directory "File Control Blocks" (FCB's) which will be cached, system wide.

The value should be set to the total number of directories likely to be active in the system at any one time.

This value should be set conservatively, as the space used is non-paged pool.

(Note -- The /ACCESSED qualifier to the MOUNT and INITIALIZE commands and the SYSGEN parameter ACP_SYSACC which, prior to VMS 4.0, controlled this cache size for ODS II disks now have no effect.)

The parameter ACP_DATACHECK is set wrong by AUTOGEN. It should be zero. There is no need for write checks on modern disk equipment.

* Can change this one. Now tech doesn't rep. it. bec. of error corrected controllers.

Implications

Change cluster size on all disks to 4 or a multiple of 4 if excess space can be tolerated. Consider devoting a disk with a large cluster size to only large files.

Excessive file functions are a user or software problem and should be addressed via system tuning only as a last resort.

I/O Operation Capacities

Disk Drive Operation Rate Limits

There are 3 steps to completing an access on disk:

Positioning -- the disk arm must be moved to the proper cylinder.

Latency -- the drive must wait for the desired sector to spin under the read/write head.

Transfer -- the data must be transferred to or from the CPU.

Positioning is the largest component in terms of time required. Due to the multi-user nature VMS and its disks and due to the fragmentation of files, is almost always required.

The transfer time varies by the size of the I/O, but is still short relative to positioning time except for unusually long transfers.

Different types of disk devices work at different speeds. The chart summarizes those speeds for devices commonly found on VAXes and computes an average total disk operation time. It also shows a maximum operational rate for the disk under normal circumstances.

If the operation rate observed for any disk regularly approaches or exceeds the indicated values there is significant contention for that disk and response degradation is probably occurring. In extreme cases, the CPU may be experiencing idle time while waiting for disk I/O. Actions should be taken to address the problem.

Usually a single application will be found which is generating excessive volumes of I/O. It should be fixed.

If it is the case that it is the sum of the demand of many users which is causing the overload, then some reallocation of the disk files across several disks will be required.

Disk Device Timing - Average VMS I/O Operations

Disk Type and Interface	Average Seek Millisec	Average Latency Millisec	Transfer Rate Mb/sec	Transfer Time For 8k Bytes Millisec	Total Access Time	Accesses Possible per Second
Fujitsu M2351 Massbus	18.7	6.3	1.8	4.5	30	34
Fujitsu M2361 Massbus	20.7	6.3	2.2	3.7	31	33
RP07	--> 31.3 <--		2.2	3.7	35	29
CDC 9771 Massbus	16.0	13.9	1.8	4.5	34	29
CDC 9772 Massbus	16.0	8.3	2.2	3.7	28	36
RM80	25	8.3	1.6	5.1	38	26
RA80/ HSC50	25	8.3	.9	9.0	42	24
RM05/RM03 CDC9766,75	25	8.3	.98	8.2	42	24
RA81/ HSC50	28	8.3	.9	9.0	46	22
RA80/ UDA50	25	8.3	.6	13.7	46	22
RA81/ UDA50	28	8.3	.6	13.7	50	20
RA60/ HSC50	41.7	8.3	.9	9.0	59	17
RA60/ UDA50	41.7	8.3	.6	13.7	63	16
RK07	--> 37.0 <--		.54	15.0	52	19
RL02	--> 55.0 <--		.51	16.0	71	14

RAxx Disks, the HSC50, and the UDA50.

A highly touted "feature" of the HSC50 is called "elevator optimization". This is an ability to reorder disk transfer requests to minimize the amount of time the disk arm spends travelling across the disk. Under normal VMS circumstances it is useful only in cases of severe disk contention where it will slightly alleviate the response degradation. For this feature to accomplish anything, there must be a queue of pending I/O requests at the drive. If there tends to be such a queue under VMS, response is usually already unacceptably bad. This feature, because it is not priority controlled, can cause undesirable results in certain circumstances.

The ability of the HSC50 and UDA50 to optimize latency delays is of some value. It can speed access by up to 10% in some cases, particularly if the file is highly fragmented. Reducing the fragmentation is a more effective procedure.

As the timing chart reveals, RAxx devices are considerably slower, particularly when on a UDA50, then when connected via Massbus interfaces. This is because, although these disks have relatively fast transfer rates, these controllers can transfer to the CPU only at considerably slower rates.

Channel Capacity

With the exception of the HSC50, disks are connected to the VAX via simplex channels. That is, the channel can only do one thing at a time -- transfer data or transfer I/O requests. (Note that as many as all disks on the channel can be positioning simultaneously, or as many as all but one with that one transferring data.) The speed of the channel (except the UDA50) is the disk transfer speed. If a channel is busy transfering more than 50% of the time, contention is probably occurring and response delays will ensue.

For example, a channel operating a set of Fujitsu M2351 ("Eagles") at 50% busy can be transferring 1600 blocks per second -- 100 average I/O's per second can be accommodated. The total disk I/O demand of typical VAX workloads (including 8x50's) seldom comes close to

these numbers and there is almost never a need to split disks among multiple Massbusses or Massbus type interfaces. The exception is with high volume I/O applications (or moderate volume, on the UDA50) which do large data transfers. Because of its inefficiencies, if RMS is used to do the I/O, it is almost impossible to exceed the capacities of Massbus type channels. (Emulex and System Industries SBI type interfaces are functionally equivalent to Massbusses.)

The CI bus is multiplexed and can do several transfers at once. Given the bus arbitration activity and inter-CPU traffic which occurs in clustered environments, predicting its data handling capacity is difficult, but is likely to fall in the 3 to 5 megabyte per second range. Many multiple 8600 clusters will find one CI inadequate, as may clusters with 5 or more 780's and 785's and large amounts of I/O processing.

VMS provides an ability for CPU's which are connected by a CI bus to do I/O on a disk locally connected to another CPU (such as with a Massbus). Doing so more than triples the CPU cost of the I/O, both in the effort needed to set it up, and the slowdown effect due to memory contention.

Implications

For fastest performance, avoid DEC disk subsystems and use Eagles or CDC XMD's with third party interfaces. VMS cluster logic works perfectly well with third party equipment with multiple CPU porting to the controllers. In addition, System Industries offers an alternative, and more efficient, clustering approach.

Fastest DEC performance is from RM80 drives. Avoid RA60's -- the positioning is too slow for most intensive applications and its high transfer rate is negated by the HSC50 or UDA50.

Respond to disk contention as a programming problem first.

Avoid allowing I/O access to devices connected locally to one CPU from another CPU over a CI bus.

Choice of Media

For high volume, sequential file applications, the most effective media is tape, both for performance and media and handling costs.

The performance of DEC layered software (RMS) for tape access is miserable. But VMS itself (QIO access) is exceptionally efficient. It is relatively easy to develop a suitable software access layer for high level applications.

DEC start/stop tape drives (e.g., the Tx78, TU77, TS11) are overpriced and too unreliable for significant general purpose tape processing. DEC streaming tape drives, on the other hand, exhibit exceptional reliability. The Tx81, because of the use of DSA architecture, can perform effectively if software capable of streaming it is used.

Disk Accessing Techniques

RMS and Additional DEC Software Access Levels

RMS tends to be inefficient for all forms of I/O. Using additional layers (high level language interfaces) adds to the inefficiency.

RMS has many options, with some (seldom the defaults) providing improved efficiency. Some high level language interfaces can operate some of the options. Under Fortran, with its data structure definition facility and the definitional libraries of RMS control structures, RMS can be called directly from Fortran with ease.

Writing efficient replacements for RMS functionality is relatively easy (except for complex indexed access), even in Fortran.

Sequential Access

If applications can be designed to work in a sequential mode, great efficiencies will be achieved. Any operation which is going to access a substantial portion of the data in a file should do so sequentially, even if additional sorting steps must be introduced.

Fixed length records are almost always more efficient than variable. Unless the wasted file space will be gross (over 50%) a fixed format should be chosen. Fixed length records have the added advantage that they may be accessed randomly.

If large sequential files are being used, a large buffer ("multiblock count") should usually be employed to minimize the number of I/O transfers. Buffer sizes of up to 65k bytes are possible, and not necessarily unreasonable.

Multiple buffers and the Read Ahead and Write Behind options are never appropriate for sequential files on multi-user systems. They waste memory and add overhead. These options are set under Fortran and other languages by default.

Under RMS, use "LOC" mode whenever possible when reading files (sequential or otherwise). This avoids moving large blocks of data from buffer areas.

Random Access

Several forms of random file access exist:

Indexed
: Powerful, but general purpose, and consequently a user of large amounts of resources (CPU time, memory and disk storage). Other, simpler techniques should be employed if possible.

Direct Access
: Sequential files with fixed length records may be accessed randomly by record number. Very efficient and quite useful if a simple count index is natural to the data or can easily be maintained. For simple key stuctures, build your own index.

Relative
: A access scheme for variable length records where RMS expands the records to fixed length and adds a data length and a data presence flag. User software can probably do a better job of these functions in almost all cases.

Access by RFA ("Record File Address")
: RMS maintains an RFA for each record in a sequential or indexed file. If these values are saved when the file is created or later accessed, any record can be efficiently retrieved from the file by specification of its RFA. (RFA's are unique over the life of an indexed file -- if a record is deleted, its RFA will never be reused.)

Accessing files randomly requires careful attention to buffer size (Multiblock Count) and the number of buffers in use (Multibuffer Count) to balance the number of physical I/O's and memory use in such a way that overall efficiency is maximized. Buffer sizes should be kept small except where there is a high likelihood of accessing sets of records which are physical neighbors of each other. Defining multiple buffers with RMS is tricky, since the user has no control over which are retained and which are overwritten. Experimentation provides the only reliable answers to whether increasing the buffer counts will improve performance.

If some form of deterministic data caching would be of use, RMS will almost always have to be avoided so that an effective scheme can be implemented.

Indexed Files

When using Prologue 3 files, be sure RMS is getting effective data compression (use ANALYZE). In critical cases, timing comparisons between Prologue 3 and 1 or 2 should be made.

Avoid multiple keys unless the file is relatively permament and not updated frequently, or at least, not during times when there are other demands for the CPU. If one of the keys is an artificially constructed "serial number", use the RFA instead. If a secondary key will only be used for sequential access, consider keeping it outside the file in a separate file with RFA's to the data file. As records are added to the file, append data to the reference file. Before using it for access, sort it.

The key which will be used most often for sequential or near sequential access should be the primary key.

When doing batch updates, they should be done in order of the primary key. Sort the update transaction data first.

If updates will be frequent, select an appropriate fill factor and introduce a regular program of file reorganization.

Bucket sizes should be started at what ANALYZE suggests, but for heavily used files, experimentation should be carried on from there. If access pattern is anything remotely near sequential (as opposed to purely random) considerably larger bucket sizes will almost always perform better. The contentions in the VMS literature that larger bucket sizes do not help unless the number of index levels are reduced, or that, at best, larger bucket sizes allow one to reduce the number of I/O's at the expense of CPU time are totally false. Generally, multiple areas should be used (at least one for each key and one for the data) so that different buffer sizes can be chosen for each.

Setting effective buffer counts requires care and experimentation. Different values should be chosen for each different type of use of a given file. For

sequential access, only 2 should be used. For random access, the effectiveness of larger numbers depends on the locality of access and the size of the file. Since RMS's caching strategy favors keeping index data in memory, few (if any) previously accessed data buffers will ever be available without redoing the I/O.

Shared Access by Multiple Processes

Read Only

Global buffers are useful if the same areas of the file are likely to be needed by simultaneous users on the same CPU. This would generally be true for indexed files, unless separate users are accessing by different keys. Global buffers should not be used with sequential access.

Read / Write

This must be avoided at all costs for high activity files if effective performance is desired. The highly generalized VMS Distributed Lock Manager consumes large amounts of CPU times -- 1 to 3 milliseconds of CPU time (780/8200) per request -- and each RMS Get or Put will typically result in more than one Lock Manger access. To this must be added the time spent by RMS to deal with the locks. If multiple CPUs are accessing the file, the time spent per lock request can increase by 200% or more.

Better strategies exist. For example, the lock manager can be bypassed and a specific control mechanism erected using shared memory and CEF's or AST's for communication. (Mailboxes should never be used in active interprocess communication situations.) Or, a single process can be designated the I/O server and requests handled with shared memory. In multiple CPU environments, the shared memory would be in an MA780, which, as an inter CPU communication device, is orders of magnitude more efficient and powerful than the CI bus.

Another strategy is to have the application handled totally by one process which supports multiple control flows. The ease of doing

this under VMS is one of its greatest features, yet one seldom sees it in use, probably because it isn't as "fancy" as other approaches.

Properly handled, approaches along the lines suggested above can be 90% more efficient for multiple access applications.

Implications

Never rely on process or system or language defaults for serious of use RMS. Even those suggested by ANALYZE must be questioned. Options must be set file by file, and for many options, by type of use for the same file.

Never turn off the deferred write option under RMS, unless reducing the risk of data corruption due to a system crash is worth a very serious performance loss. (Note that this feature does not absolutely insure against corruption.) Be aware that it is off by default in some languages (COBOL).

} check this! - in open statement literature.

To reliably determine the effect of an RMS parameter or structural change, comparisons of the actual use of the software is vital. It is very difficult in any synthetic testing to simulate the access patterns and load conditions experienced in actual operation. One should make a change, test for correctness and not unacceptable performance, then install the changed version as the production system. Assemble performance data from before and after the change and compare.

Database Management Systems

These systems (should):

Lower development time and labor (high level functionality)

Allow applications to change with less software revision effort.

Their drawback is exceptionally large resource consumption.

Questions which must be asked before deciding to employ them (and which seldom are):

Are the development savings (if any!) worth the performance cost?

Does the package force pounding square pegs into round holes and is it worth the less effective solution which ensues?

Is the application really subject to frequent change? Can the nature of the expected changes be forecast and appropriate flexibilty built into a lower level solution?

