A general description of the Virtual Storage Access Method (VSAM) is followed by a qualitative discussion of performance expectations. VSAM data-set design parameters are discussed with respect to performance tradeoffs. Analytic techniques are developed for relating some of the VSAM performance sensitivities to data set design parameters.

VSAM data set design parameters

by D. G. Keehn and J. O. Lacy

The Virtual Storage Access Method (VSAM)[1] has been developed for use with virtual storage operating systems. VSAM grew out of the need for an access method that allows data to be accessed both directly by key and sequentially in key-defined collating order. Conventional index-sequential access methods that satisfy this need usually use a chaining technique to insert additions into a file after it has been initially loaded. With these techniques, performance degrades rather substantially as more and more additions are made. VSAM has been designed to avoid performance degradation while retaining the index-sequential facility. Two new logical concepts defined in VSAM are used to manage the space associated with data: the Control Area (*CA*); and the Control Interval (*CINV*). An index is used to address the records contained in control areas and control intervals. An insertion technique is used that works well even after the file has had many records added. The result is a sequential direct insertion facility that—compared with conventional chaining techniques—performs well and continues to do so as the file is built up. Although VSAM has been designed for use with virtual storage operating systems, it may also be used with all of the OS/370 operating systems.

Our purpose here is to provide concepts to consider when designing a VSAM data set. This paper describes VSAM and then uses that description to make some qualitative statements about VSAM performance expectations as compared with other methods. Some performance tradeoffs are discussed with respect to VSAM data set design parameters. Finally, analytic techniques for some of the crucial VSAM performance effects are devel-

Figure 1 Control area (CA) with sequence sets and control intervals (CINVs)

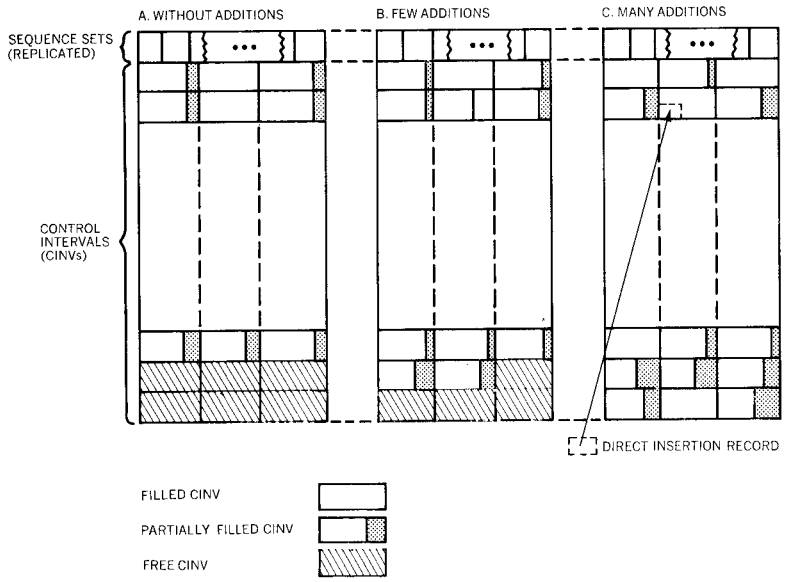
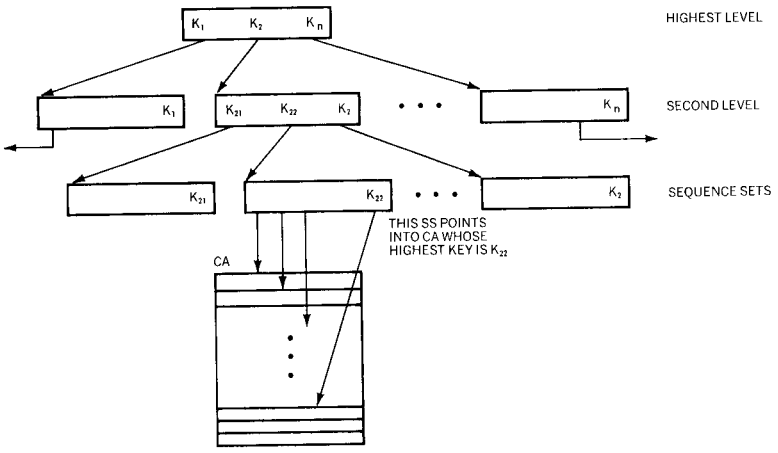


Figure 2 Three-level VSAM index



oped to help the reader appreciate the VSAM performance sensitivities and to lead the way toward a more detailed understanding of VSAM and its performance.

The concepts presented here grew out of an effort that resulted in an APL program to aid VSAM data set design and analysis. As a consequence of that effort, the analytic techniques have been applied to some real VSAM data sets, and results of those analyses have been spot checked against detailed measurements of those same data sets.

data set structure

VSAM data sets are constructed either as Key Sequenced Data Sets (KSDS) or Entry Sequenced Data Sets (ESDS). ESDS are sequential add-on data sets with no index structure. Records in KSDS have a key embedded in each record that defines a collating order for the records. The records are initially loaded in the defined collating order and continue to be accessible in that order (known as sequential access) as new records are inserted. An index is provided for keyed direct operations, that is, not in a predetermined order.

A VSAM data set consists of a number of Control Areas (*CA*), each of which consists of a number of Control Intervals (*CINV*), which, in turn, consist of a number of records. The records within a *CINV* are physically maintained in sequence according to the key embedded in each record. Each *CINV* also contains control information regarding where each record starts within the *CINV* (so as to allow variable length records). A *CA* is often a cylinder (but may be specified to be smaller) of a Direct Access Storage Device (DASD). Figure 1 represents three instances of the same VSAM *CA* as situated on a *DASD* cylinder. (The first track of the *CA* is occupied by a sequence set that is to be described later in this paper.) Each of the other tracks contains three *CINVs*. The shaded areas represent free space.

index

A VSAM Key Sequenced Data Set (KSDS) has an index for direct operations. Many index entries are stored together on DASD, and they are accessed in blocked units. Each index record is a key-pointer pair where they key is the highest key in the pointed-to block, and where the pointed-to block is another index block or Control Interval (*CINV*) in the data set. The blocks make up levels where the highest level consists of a single block, and each of the lower levels consists of the blocks pointed to by the next higher level. This may be thought of as a fan-out effect. Each of the blocks in the lowest level of the index (called the Sequence Set) addresses a particular Control Area (*CA*), and resolves a key to the *CINV* in which the desired record resides (if it exists). Figure 2 represents a fan-out, in tree form for a three-level VSAM index.

A record is located via this index tree by using a less-than-or-equal comparison on each level until the appropriate *CINV* is located. Optionally, the Sequence Set can be stored with the data it addresses. If this is the case, the Sequence set occupies the first track of the *CA*, and is replicated as many times as will fit on the track (to save rotational delay—*latency*—waiting for the beginning of the desired record to come under the read/write head). Higher levels of the index may also be replicated. VSAM compresses the key entries in the index so that redundant key

information is removed, which also helps to save latency. It is thus possible for relevant information to be stored in smaller blocks, which results in more (replicated) blocks per track. Another effect of key compression is to increase the number of pointers in each index block, which, in turn, reduces the number of levels in the index component of the KSDS. Figure 1 shows the placement of the VSAM Sequence Set on the first track of the cylinder. Index design is discussed in greater detail in Reference 2.

distributed free space

As a VSAM file is loaded, space is left free for future additions. The space to be left free is specified by the following two parameters: (1) the percentage of each loaded *CINV* to be left free; and (2) the percentage of each *CA* to be left free. Each *CINV* is loaded until it is left with at least that percentage of free space, and each *CA* is loaded until it is left with at most the specified percentage of free *CINVs*. Thus, before any records are inserted, each *CA* (except possibly the last) is approximately uniformly loaded to a certain percentage of fullness. The free *CINVs* make up a pool that provides *CINVs* for records as *CINVs* become full and split as described later in relation to the algorithm for making insertions. The shaded areas in Figure 1 represent both forms of free space—the free space within *CINVs*, and the free *CINVs* at the bottom of the *CAs*.

direct operations

VSAM provides keyed direct operations and data sets by using the key-index structure just described. Records may be retrieved and/or updated by key (and, hence, in any order), and new records may be inserted in the proper place (according to their key) in the data set.

An algorithm for keyed direct retrieval and the updating of data sets is detailed as follows:

1. Read (from DASD) and search the high level(s) of the index by using the key of the desired record. For the high levels of the index already in main storage, it is not necessary to retrieve them from DASD.
2. Read and search the sequence set block. The result is the DASD address of the *CINV* in which the record is stored in the file.
3. Read that *CINV*. Using the control information stored in the *CINV*, find and return the desired record to the user. If the request is for a retrieval, the request is complete.
4. If the user returns an updated record, VSAM replaces the corresponding record in the *CINV*. Rewrite the *CINV* onto DASD. The direct update request is complete.

insertion

The VSAM insertion algorithm is designed so that a new record is stored with records that are close to it in the collating se-

quence. This effect can be accomplished because of the free space allocated when the file is loaded. Usually, an insertion is performed as a direct update, since the appropriate *CINV* is read. The record is inserted into the *CINV*, and the *CINV* is rewritten. However, if the free space still in the *CINV* is not large enough to accommodate the new record, the *CINV* is split into two new *CINV*s.

The insertion algorithm is given as follows:

1. Read and search the high levels of the index, using the key of the new record. If some of the high levels of the index are already in main storage, it is not necessary to read them.
2. Read and search the sequence set block. The result is the DASD address of the *CINV* in which the record would be stored if it were already in the file.
3. Read that *CINV*. If there is not enough free space in the *CINV* for the new record, go to step 5. If there is enough free space, insert the record into the *CINV* where it logically belongs, moving records as needed so that within the *CINV* the records are physically collated. Note that the highest key in this *CINV* does not change, and, hence, the sequence set does not need to be changed.
4. Rewrite the *CINV* onto DASD. The record is now inserted into the file.
5. Split the *CINV*. If there are no free *CINV*s in the *CA* (as determined from the sequence set), go to step 7. If there are free *CINV*s, build two new *CINV*s in storage from the old *CINV* and the new record. Write both the *CINV*s onto DASD. This process (including step 6) is called *splitting a CINV*.
6. Update the sequence set to reflect the old and new *CINV*s in the collating sequence. Rewrite the sequence set onto *DASD*. The new record is now inserted into the file.
7. Split the *CA*. If no free *CINV*s remain in the *CA*, the *CA* must be split, which involves allocating a new *CA* from DASD and writing half the *CINV*s into the new *CA*.

Figure 1B shows a VSAM-inserted *CA* with some split *CINV*s. Figure 1C shows a *CA* that is about to split.

sequential operations

Sequential operations are involved in the processing of records, one by one, in the order defined by the collating order of the embedded keys. (In OS/VS VSAM, the sequential retrieval of records is accomplished by retrieving the *CINV*s in sequence, and by using the sequence set as a guide to *CINV* splitting.) Usually, *CINV*s are transferred to and from main storage in groups so that each access to DASD is amortized over many records. Additionally, if the user has allocated four or more buffers, half are filled at a time to overlap the DASD access time with the user processing time.

Performance implication

The VSAM design offers some significant methods for improving performance over other access methodologies. The new methods are explored in this section, by using comparisons with current index-sequential procedures.

The VSAM index structure has several higher index levels that resolve to a Control Area (*CA*) index (Sequence Set). The Sequence Set always resolves to the desired Control Interval (*CINV*), because—when a *CINV* is split—the Sequence Set is modified to reflect the split.

index

Index sequential (ISAM) methods use a master index, a cylinder index, and a track index. Even after resolving to a DASD track, however, the track must still be searched, with the further possibility of having to follow an overflow chain to retrieve the desired record. In that method, the various indexes themselves never need to be updated, whereas VSAM must continue to update the index blocks as splits occur. The ISAM method amounts to spreading the pointing information over the disk surface, whereas VSAM keeps all the pointers compacted in one index block (at the expense of continued updating).

distributed free space

Access methods that allow for inserts often have a form of distributed free space. The VSAM scheme leaves a specified percentage of free space in each *CINV* and a (possibly different) specified percentage of free space in each *CA*. Therefore, the first addition goes into the appropriate *CINV*, and, with a lightly inserted file, an addition to a VSAM file is much like a direct update. Previous methods provide free space only at physical boundaries, which implies that additions go immediately into an overflow area. Chaining is usually used to locate records that have been inserted.

The VSAM free-space strategy allows additions to be made to the expected *CINV*. When an overflow occurs, the *CINV*s are split and the sequence set is updated, with the result that the insertion technique works well even as the file continues to be added to. ISAM methods use chaining to overflow records so that, even with the first few additions, the insertion performance begins to degrade.

blocks versus CINVs

VSAM allows some free space to be left in each *CINV* as it is loaded. Hence, at zero or light insertion levels, the VSAM sequential performance may suffer compared to a fully utilized block. However, the splitting strategy results in a continued high blocking factor, as compared to the overflow chain strategy, which results in a diminishingly effective blocking factor as additions are made. As heavy addition levels begin, *CA* splitting adds

*CA*s to the file and results in longer seeks for direct operations. An independent overflow area, as used with the overflow chain strategy, results in extra seeks between the prime and overflow areas at heavy addition levels.

Before delving into more detailed performance measuring techniques in the following sections, it might be helpful to make a few prelimary qualitative statements about performance, which are based on the more detailed material. In practice, this level is often adequate, because an understanding of the key design dependencies is usually sufficient to make good design choices for VSAM.

Control interval size. The amount of DASD space required by a Key Sequenced Data Set (KSDS) and the sequential request times are sensitive to the choice of *CINV* size. A larger *CINV* size produces better sequential performance. It is possible, however, to waste DASD space with certain combinations of *CINV* size and record size.

Distributed free space. When designing for random direct additions, choose the free space in a *CINV* larger than the free space in a *CA*. The splitting of *CA*s is better controlled by free space within the *CINV* than free *CINV*s with a *CA*. Make the free space within a *CINV* at least as large as the design percentage of insertations so that the *CA* splitting is small. To avoid degradation of sequential operations, sufficient free space must be specified. However, too much free space in a *CINV* yields a poorly utilized block for transferring data to and from DASD.

Index buffers. Buffers for holding the *CINV* and *CA* indexes in main storage are not analyzed in this paper, but they are included here because of the importance of direct operations. Therefore, include some extra index buffers when direct operations are significant for an application. In a purely sequential application, two index buffers are usually sufficient. Allow at least one more index buffer than the number of levels in the index for direct applications. These index buffers allow VSAM to keep the higher levels of the index in main storage, thus reducing the number of DASD accesses to the index component.

Data buffers. The number of buffers for holding data in main storage have a significant impact on the performance of sequential operations. For applications with direct operations only, more than two data buffers add little to performance. In general, as the number of data buffers increases, sequential performance should improve. A point that should be kept in mind regarding direct operations with keys that are clustered together is that VSAM usually checks the current data buffers to determine

whether the requested record is present. This may reduce the number of DASD accesses required for an application.

Allocation unit. This unit is a parameter that should be so chosen that VSAM allocates a cylinder for a *CA*. If allocation units are specified in terms of records or tracks, a *CA* of less than a cylinder can result. When a *CA* is less than a cylinder, additional device interruptions must be handled by VSAM on sequential operations.

Index options. VSAM provides four choices of index component. One may choose to embed the Sequence Set with the data or not and choose to replicate the nonembedded part of the index or not. Embedding the Sequence Set can reduce direct retrieval and insertion times.

Analytic modeling techniques

We now discuss several analytic modeling methods that are useful for understanding the performance of VSAM. The analysis applies to single-string requests. These are repeated requests of the same kind from a single task such that each request is synchronous with the completion of the prior request in the repeated pattern of requests. A stochastic model is first established for the distribution of split *CINV*s and *CA*s in a VSAM Key Sequenced Data Set (KSDS). Given the model, we can then analyze the DASD space required at various insert levels as well as the sequential and direct single-string access times.

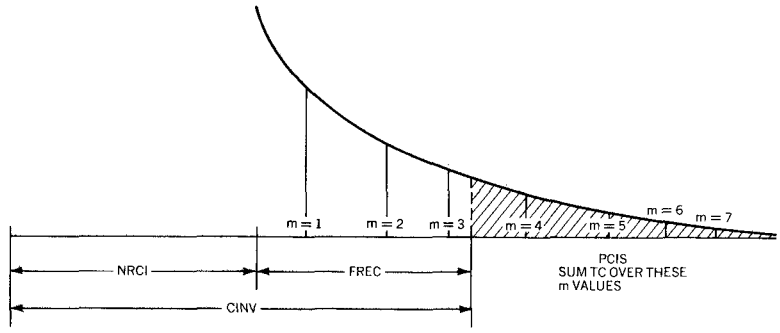
direct insertion

We first account for *CINV* and *CA* splitting as a result of direct additions. From the discussion on making insertions, recall that when the free space in a *CINV* is not able to accommodate the addition at hand, a split *CINV* results.

In the analysis, it is necessary first to compute the probability of a *CINV* splitting. We begin by defining *REC* as the number of records in the KSDS at loading time. The keys of these records are modeled as random numbers chosen from a distribution of key values denoted by F_K. The loading process puts *NRCI* fixed-length records into each *CINV*. Subsequent to the loading process, I random additions with keys chosen from F_K are made. If m denotes the number of additions falling in any particular load time *CINV*, then the probability of exactly m additions in an interval is given as follows:

$$\pi\{m/I, REC, NRCI\} = \frac{\binom{I}{m} \cdot \binom{REC}{NRCI} \cdot \frac{NRCI}{NRCI + m}}{\binom{I + REC}{m + NRCI}} \qquad (1)$$

Figure 3 Range of summation of probability of at least one CINV splitting



where

$0 \leq m \leq I$

The probability of at least one *CINV* split is denoted as *PCIS* (*REC*, *NRCI*, *FREC*), and is given by summing Equation 1 over the number of records that overflow a loaded *CINV*. If *FREC* denotes the number of free records available in a *CINV* after loading, then

$$PCIS\ (REC,\ NRCI,\ FREC) = \sum \pi\ \{m/I,\ REC,\ NRCI\} \qquad (2)$$

where the summation is over all m such that $FREC + 1 \leq m \leq I$

For purposes of data set design, a primary interest is the first *CINV* split after loading. In evaluating Equation 2, only the leading terms of the sum are significant. Figure 3 shows the relationship between *NRCI*, *FREC*, and *PCIS*. It should be noted here that $\pi\{m/I,\ REC,\ NRCI\}$ and, hence, *PCIS* (*REC*, *NRCI*, *FREC*) does not depend on the particular key distribution F_K. This fact allows a simple analysis for any monotonic F_K.

The analysis presented here accounts for two sources of nonuniform distribution of records across the *CINV*s. The first source occurs at loading time, when the finite number of records in each *CINV* results in an uneven distribution of probability values contained in each *CINV*. Ideally, if the keys of the loaded data set were perfectly representative of the distribution F_K, then each *CINV* contains equal probabilities of having subsequent insertions fall in that *CINV*. The model shows this not to be the case. Rather, this probability is itself a random variable and its distribution is of interest. The second source of nonuniformity occurs when insertions are made. The small probability held by each *CINV* implies a large variation in the number of inserted records across the *CINV*s. For direct insertions, we do not expect each *CINV* to fill up uniformly.

control interval probability

We now analyze the loading of a KSDS data set, and make computations to determine the distribution of the probability held by each *CINV* after the loading process. At loading time, the collection of records with keys sampled from F_K are available, so that we have the following relationships:

$$\text{Data set keys} = \{X_1, X_2, \cdots, X_{REC}\}$$

where

$$\Pr\{X_\alpha \leq t\} = F_K(t)$$

for each $\alpha \varepsilon (1, 2, \cdots, REC)$

These records are sorted by key value and grouped into *CINV*s with *NRCI* records in each *CINV*. The key range assigned to the *i*th *CINV* by the loading process is given by the random interval,

$$\{\text{Key values in the } i\text{th } CINV\} = (X^*_{(i-1)NRCI}, X^*_{iNRCI})$$

where the $\{X_\alpha{}^*\}$ are the result of sorting $\{X_\alpha\}$, $\alpha \varepsilon (1, \cdots, REC)$

This notation leads to asking what the probability is that, subsequent to the loading process, a given insert will be placed in the *i*th *CINV*. That is, it is necessary to compute the probability density of P_i where

$$\Pr\{Y \varepsilon (X^*_{(i-1)NRCI}, X^*_{iNRCI})\} = P_i \tag{3}$$

and where the inserted key Y also has the probability distribution F_K. With this last fact in mind, Equation 3 may be rewritten as follows:

$$P_i = F_K(X^*_{iNRCI}) - F_K\,(X^*_{(i-1)NRCI}) \tag{4}$$

where

$$1 \leq i \leq REC/NRCI$$

Thus the random variables P_i depend on the sorted key values through the cumulative distribution function F_K.

Equation 4 appears to be complicated, but it is easily solved because of the following two facts:

(a) The sorted distribution of keys $F_K\,(X_\alpha{}^*)$ has the same probability law as an ordered sample of size *REC* from a uniform probability distribution on (0, 1). This follows from the fact that the unsorted random variables $F_K\,(X_i)$ are distributed as follows:

$$\Pr\{F_K(X_i) \leq t\} = t$$

where

$$0 \leq t \leq 1$$

(b) Also, all the probabilities P_i have the same first order probability distribution. In particular, any P_i is distributed as the first as follows:

$$P_1 = F_K(X^*_{NRCI})$$

To show (a), we note the following conditions placed on F_k:

$$\Pr\{F_K(X_i) \leq t\} = \Pr\{F_K^{-1}F_K(X_i) \leq F_K^{-1}(t)\}$$
$$= \Pr\{X_i \leq F_K^{-1}(t)\} = t$$

where

$$0 \leq t \leq 1$$

The conclusion in (a) follows, since sorting the $\{X_i\}$ and then F_K gives the same result as ordering the F_K $\{X_i\}$.

At this point, assume the conclusion for fact (b) and refer to the Appendix for a proof.

Combining the results of (a) and (b), the distribution of P_i can be completed by finding the distribution of the $NRCI$th largest out of REC uniformly distributed random variables on the interval (0, 1). Let $Z_1, Z_2, \cdots, Z_{REC}$ denote uniformly distributed random variables on (0.1), and note that the $NRCI$th term in an ordered sequence is less than t if and only if there are least $NRCI$ out of REC keys that are less than or equal to t.

$$\Pr\{P_i \leq t\} = \Pr\{\text{at least } NRCI \text{ of } Z_1, Z_2, \cdots, Z_{REC} \leq t\} \qquad (5)$$

Since the Z are independent, the computation in Equation 5 is given by the following binomial distribution:

$$\Pr\{P_i \leq t\} = \sum_{j=NRCI}^{REC} \binom{REC}{j} t^j (1-t)^{REC-j}$$

Differentiating this expression and evaluating it at $t = p$ yields the following desired probability density for Equation 3:

$$\frac{d}{dt}\Pr\{P_i \leq t\}_{t=p}$$

$$= \frac{REC!}{(NRCI-1)!(REC-NRCI)!} p^{NRCI-1}(1-p)^{REC-NRCI} \qquad (6)$$

probability of a control interval splitting

The derivation of Equation 1 and, hence, the evaluation of Equation 2 can easily be obtained from the results of Equation 6. Suppose the probability p contained in a $CINV$ is known. The probability that m out of I additions fall into the key range spanned by this $CINV$ is given by the binomial law as follows:

$$\Pr\{m \text{ inserts in a } CINV/p\} = \binom{I}{m} p^m q^{I-m} \qquad (7)$$

where

$$p + q = 1$$

Equation 7 is the conditional probability of the additions falling within *CINV* when p is known. Averaging Equation 7 over the distribution on p values given by Equation 6 yields the probability of m insertions falling in *CINV* as follows:

$$\Pr\{m \text{ inserts in a } CINV\} =$$

$$\binom{I}{m} \int_0^1 p^m q^{I-m} \frac{REC!}{(NRCI-1)!(REC-NRCI)!} p^{NRCI-1} q^{REC-NRCI} dp \tag{8}$$

The result of Equation 1 is a result of Equation 8 and the definition of the beta function as given by Feller.[3]

probability of a control area splitting

The techniques of finding the probability of a *CA* splitting parallel those used for computing *PCIS*. The reason for this is the similarity of the manner in which free space is allocated with a *CINV* and a *CA*. Figure 1A shows a *CA* loaded with free space in *CINV*s and free *CINV*s within the *CA*. As additions split a *CINV*, a free *CINV* is used, and eventually the free *CINV* pool is depleted. Figure 1B shows a *CA* with some split *CINV*s, and Figure 1C shows a *CA* with the free *CINV* pool depleted. For direct additions with random keys, only the originally loaded *CINV*s contribute to the number of splittings because the probability of the double splitting can be neglected. With this approximation noted, we have the following expression for the probability of a control area splitting:

$$\Pr\{CA \text{ splitting}\} =$$
$$\Pr\{\text{number of loaded } CINV \text{ splittings exceeds the number of 3 } CINV\text{s within a } CA\}$$

Each originally loaded *CINV* is a source of *CINV* splittings with probability *PCIS*. Let *CIPCA* denote the number of nonempty *CINV*s per *CA* after loading. Taking the contribution from each load time *CINV* as statistically independent, the probability of a splitting is given by summing the tail of the following binomial distribution:

$$PCAS = \Pr\{\text{Control Area splitting}\}$$
$$= \sum_{\substack{m=1+\text{free} \\ CI \text{ per } CA}}^{CIPCA} \binom{CIPCA}{m} PCIS^m QCIS^{CIPCA-m} \tag{9}$$

Here, $QCIS + PCIS = 1$, and free *CINV*s denote the free *CINV*s within a *CA*. For purposes of computation, the binomial law in Equation 9 can be replaced by the Poisson law when *CIPCA* is large (e.g., exceeds 20) and *PCIS* is small. The parameter in the Poisson law is given by the product $CIPCA \times PCIS$.

At this point a technique for computing *PCIS* and *PCAS* has been established, given certain data set parameters. It is still neces-

sary to relate the external data set characteristics to the parameters available to the data set designer. With this relationship established, the DASD requirements can be computed as a function of the data set design parameters.

data set design parameters

This section gives the facts necessary for relating external data set characteristics to the variables of the direct insertion model. The following list represents the parameters needed to define a VSAM data set.

- *Data control interval size* (*CNV*) must be $i \times 2^k$, where $k \leq 9$.
- *Physical block size* (*PBS*) is chosen by *VSAM* to be one of {512, 1024, 2048, 4096}, whichever is the largest divisor of *CNV*.
- *Free space parameter FCI* is the percentage of free space in a *CINV*, and *FCA* is the percentage of free *CINV*s in a *CA*.
- *Index options* specify whether to imbed the sequence set and whether to replicate the nonimbedded part of the index. See Table 1 for a list of possible combinations and numerical settings for the *IOP* parameter.
- *Number of records* in the data set at load time is *REC*.
- *Average logical record size* in bytes is *EXSZ*. Only fixed-length records are considered; the maximum record size must fit within one *CINV*.
- *Space allocation units* for the data component are specified by cylinder, track, or record.
- *Key length* is *KEYLEN*. Also specify and estimate the compressed key size (*CKS*).

DASD space computation algorithm

The following algorithmic steps are a reasonably accurate representation of the computations necessary to load a VSAM data set. Throughout this algorithm, the APL language is used to express the algebraic operations.

1. Number of records in a data *CINV* (*NRCI*). For fixed-length records, an overhead of 10 bytes per *CINV* is required. The free space parameter *FCI* specifies the minimum amount of free space in the *CINV*. Compute first

$$NRCI \leftarrow \lfloor ((-10) + (1 - FCI) \times CNV) \div EXSZ \qquad (10)$$

2. Maximum number of records in a *CINV* (*MRCI*). When all the free space in a *CINV* is filled with records, compute *MRCI* as a preliminary step to computing *PCIS*

$$MRCI \leftarrow \lfloor ((-10) + CNV) \div EXSZ$$

3. Number of data *CINV*s per track (*CIPTRK*). Compute *CIPTRK*

from the physical blocks per track (*PBTRK*) and *CINV* size (*CNV*).

$$CIPTRK \leftarrow PBTRK \div CNV \div PBS$$

All terms are defined except *PBTRK*, which is computed from device characteristics as follows:

$$PBTRK \leftarrow 1 + \lfloor ((\text{track capacity-last block}) \div (\text{length of other blocks})$$

In applying this formula, use the DASD overhead values without keys. The explicit values for such terms as track capacity depend on the device type.

4. Information regarding the units of space allocation for the data component, which is a key variable throughout this algorithm, is the number of tracks in a *CA* (*TRPCA*). The algorithm starts by assigning a value to *TRPCA*, but later this variable may be decreased to satisfy certain index component constraints. There are three ways to assign *TRPCA*.

   If space is specified in cylinders and the sequence set is adjacent to the data, assign tracks per cylinder less one, otherwise take tracks per cylinder.

   $$TRPCA \leftarrow \text{tracks per cylinder} - IOP \leq 2$$

   If space is specified in units of tracks, take the smallest number of primary tracks, secondary tracks, and tracks per cylinder as *TRPCA*. Adjust this value by subtracting one track if $IOP \leq 2$.

   If space is specified in units of records, convert the primary and secondary record values to tracks as follows, using $\lceil$, the ceiling function:

   $$\textit{Primary tracks} \leftarrow \lceil EXSZ \times \textit{primary records} \div (PBS \times PBTRK);$$

   $$\textit{Secondary tracks} \leftarrow \lceil EXSZ \times \textit{secondary records} \div (PBS \times PBTRK).$$

   Apply the space specified in units of tracks to compute *TRPCA* following the procedure as though it had been originally specified in units of tracks.

5. Maximum number of *CINV*s per *CA* (*MCIPCA*) is computed as follows:

   $$MCIPCA \leftarrow \lfloor CIPTRK \times TRPCA$$

   Since a *CINV* cannot span a *CA*, the floor function $\lfloor$ is applied to the product of *CINV*s per track and tracks per *CA*. Note that all the *CINV*s in a *CA* are pointed to by a sequence

set entry. This fact may require a change to this value farther on in the algorithm.

6. Loading time records per *CA* (*RECPCA*) is a quantity that is basic to computing the number of tracks required for the data component at loading time as well as after inserts, and is computed as follows:

$$RECPCA \leftarrow NRCI \times (MCIPCA - \lfloor FCA \times MCIPCA)$$

This equation states that the number of records in a *CA* at loading time is the product of the number of records loaded per *CINV* with the number of loaded *CINV*s in each *CA*. Note that the product $FCA \times MCIPCA$ is rounded down, which indicates that the maximum percentage of free *CINV*s in a *CA* is bounded by *FCA*. This is in contrast to the impact of *FCI* on the percentage of free space in a *CINV*. An examination of Equation 10 shows that *FCI* is the minimum amount of free space within a *CINV*.

7. Number of tracks required by the data component is computed. Consider the following zero-insertion case first:

$$\begin{pmatrix}\text{Tracks for the data}\\ \text{component of } KSDS\end{pmatrix} \leftarrow (TRPCA + IOP \leq 2) \times \lceil (REC \div RECPCA) \quad (11)$$

The factor on the far right is just the total records divided by the loading time records per *CA* from step 6. This factor is multiplied by the number of tracks per *CA* as computed in step 4. If the sequence set is imbedded with the data, the track for it is allocated from the data component.

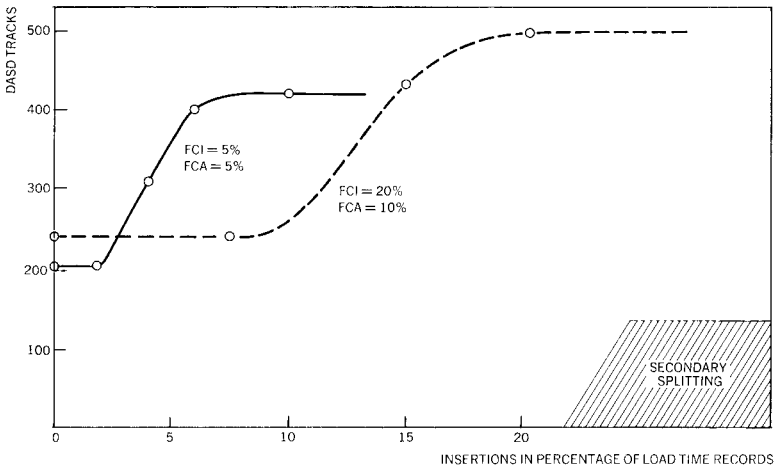
To obtain the number of tracks in the data component after inserting, note that each loading time *CA* contributes either one (if there is no splitting) or two (if *CA* is split) to the total required number of tracks. Neglect double splittings because they have a small probability over the range of insertion levels investigated here. The average number of *CA*s per loading time *CA* is computed as follows:

$$\begin{pmatrix}\text{Average number}\\ \text{of } CA\text{s per}\\ \text{loading time } CA\end{pmatrix} \leftarrow (1 - PCAS) + 2 \times PCAS = 1 + PCAS \quad (12)$$

For the range of insertion levels of interest, multiply Equations 11 and 12 to compute the number of tracks per data component at insertion level *INS* as follows:

$$\begin{pmatrix}\text{Tracks for the}\\ \text{data component}\end{pmatrix} \leftarrow (TRPCA + IOP \leq 2) \times \lceil (REC \div RECPCA) \times (1 + PCAS) \quad (13)$$

Figure 4 Expected DASD tracks for data component of VSAM KSDS



We digress from the description of the DASD space computation algorithm to study Equation 13 as a function of *FCI*, *FCA*, and *INS*. To do this, it is necessary to relate Equations 2, 9, and 13 to the data set design parameters. Starting with Equation 2, all the terms have been defined except for *FREC*, which is given by the difference $MRCI - NRCI$. (*NRCI* is given by Equation 10.) The free *CINV*s in a *CA* and *CIPCA* need to be specified to allow evaluation of Equation 9. *CIPCA* is computed as the number of loaded *CINV*s as follows:

$$CIPCA \leftarrow MCIPCA - \lfloor FCA \times MCIPCA \tag{14}$$

Finally, the free *CINV*s in a *CA* are given by the following equation:

$$(\text{Free } CINVs) \leftarrow FCA \times MCIPCA \tag{15}$$

All the terms in Equation 13 are now related to the data set design parameters.

Figure 4 shows the effect of two different choices of free space parameters on the direct access storage required by a VSAM KSDS. For a given design, the curve of DASD tracks versus insertion level is rather steep. For $FCI = 20\%$ and $FCA = 10\%$ design, *CA* splitting occurs after the ten to twelve percent insertion level has passed. We would then expect a rapid expansion of the space requirements after some design point insertion level has been exceeded. The reason for the expansion in DASD space is *CA* splitting which, for random keys, tends to occur in a band of insertion levels.

In the example, most *CA*s split between the ten and fifteen percent addition levels. Note that the DASD track curve

looks quite different for the $FCI = 5\%$ and $FCA = 5\%$ design. The addition level at which *CA* splitting occurs is much smaller, and the space required at loading time is also smaller. Hence, a performance-space tradeoff occurs between these designs. The design with the greater free space allows a large number of insertions before *CA* splitting occurs. The $FCI = 5\%$ and $FCA = 5\%$ requires less space, but *CA* splitting starts occurring at around the four percent insertion level.

We now resume the DASD computation algorithm. This part of the algorithm is used to compute the number of index *CINV*s required to point to the data component. The procedure starts with the smallest allowable index *CINV* size (512) and with a test to determine whether two basic constraints are satisfied (Equations 16 and 17). If the value for the index *CINV* is too small, the algorithm chooses the next largest value for index *CINV* size. If the allowable values of index *CINV* size are exhausted, the algorithm reduces the number of tracks in a *CA* (*TRPCA*) until the constraints are satisfied. If this last action is taken, it is necessary to recompute tracks for the data component starting at step 5 in which *TRPCA* is first used.

8. The Index Control Interval Size (*ICNVS*) has certain constraints. The index *CINV* must exceed a minimum byte value as determined by X and Y defined in the following equations:

$$X \leftarrow 31 + (3 \times MCIPCA) + 2 \times KEYLEN + 2 \qquad (16)$$

$$Y \leftarrow 31 + ((5 + CKS) \times MCIPCA) + 2 \times MCIPCA * 0.5 \qquad (17)$$

Equations 16 and 17 account for the space required for index entries in the index *CINV*. Note that *MCIPCA* is usually the dominant variable in setting the lower bound on *ICNVS*.

9. Selecting *ICNVS*. The following equation sets *ICNVS* to the smallest allowable value that satisfies the constraints required by Equations 16 and 17:

$$ICNVS \leftarrow 2*13 - +/[2]\ (Y \lceil X) \leq (512\ 1024\ 2048\ 4096)$$

If *ICNVS* is less than or equal to 4096, skip from step 10 and go to step 11.

10. Decrease *TRPCA* in steps of one track until Equations 16 and 17 are satisfied with $ICNVS = 4096$. Go back and recompute the number of tracks in the data component, beginning at step 5, using the new value of *TRPCA*.

Table 1 Index options related to the equation for tracks required by index component.

Sequence set adjacent to data	*Replicate higher level index*	*IOP value*	*Equation for tracks for index component*
Yes	Yes	1	18
Yes	No	2	19
No	No	3	20
No	Yes	4	21

11. The number of *CINV*s in the sequence set is just equal to the number of *CA*s as shown in the following equation:

$$\begin{pmatrix}\text{Sequence set}\\ CINVs\end{pmatrix} \leftarrow \lceil REC \div RECPCA$$

12. Higher level index *CINV*s are used for the remaining levels of the index. Two quantities are needed to give the number of *CINV*s at higher levels: the maximum number of *CINV*s on level i that can be pointed to by one index on the next higher level $i+1$; and the number of *CINV*s on level i.

    For $i=1$ this number is the number of sequence set *CINV*s.

    Compute the maximum number of *CINV*s on level i that can be pointed to by one index on the next higher level by solving constraint Equation 16 and 17 in the reverse direction. Let m denote the maximum number of pointers in an index. Then $m \leftarrow m_1 \llcorner m_2$ where we choose m_1 and m_2 to be the largest integers to satisfy the following equations:

    $$ICNVS \geq 31 + (3 \times m_1) + 2 \times (KEYLEN + 2)$$

    $$ICNVS \geq 31 + ((5 + CKS) \times m_2) + 2 \times m_2 * 0.5$$

    The Compressed Key Size *CKS* has been observed to be larger in the second and higher levels of the index when compared with the sequence set entries. This model requires one to estimate *CKS* from the key structure of an application. *CKS* of approximately three bytes has been measured for many sets of artificially generated keys at the sequence set level.

13. Compute the number of second-level index *CINV*s as follows:

$$\begin{pmatrix}CINVs\text{ on}\\ \text{second level}\end{pmatrix} \leftarrow \lceil (\text{sequence set } CINVs) \div m$$

    where m is computed in step 12.

14. The number of third and higher level index blocks is computed as follows:

$$\begin{pmatrix} CINVs \text{ on} \\ \text{third level} \end{pmatrix} \leftarrow \lceil (CINVs \text{ on second}) \div m$$

This process is continued by dividing the number of index *CINV*s on the current level, and rounding up until we have just one highest level index *CINV*.

15. The total number of *CINV*s in higher level indexes (*HLI CINV*s) is computed by summing over the number of index *CINV*s generated in steps 13 and 14. This term is useful for considering the index options specified by *IOP*. Table 1 shows the relations between index options and the requisite equation for computing tracks for the index.

Depending on the index option selected, the following equations apply:

$$\begin{pmatrix} \text{Track for higher} \\ \text{level index} \end{pmatrix} \leftarrow \begin{pmatrix} \text{number of} \\ HLI\ CINVs \end{pmatrix} \tag{18}$$

$$\begin{pmatrix} \text{Tracks for higher} \\ \text{level index} \end{pmatrix} \leftarrow \left\lceil \begin{pmatrix} \text{number of} \\ HLI\ CINVs \end{pmatrix} \div \begin{pmatrix} \text{Index } CINVs \\ \text{per track} \end{pmatrix} \right. \tag{19}$$

In the case of Equation 18, the sequence set is allocated from the data component and each higher level (*HLI*) index *CINV* uses one track.

Equation 19 requires a computation of the number of index *CINV*s that fit on one track. Finally, the following computation with physical block size equal to *ICNVS* is required. This equation applies when no replication is requested:

$$\begin{pmatrix} \text{Tracks for} \\ \text{whole index} \end{pmatrix} \leftarrow \left\lceil \frac{\begin{pmatrix} \text{number of} \\ HLI\ CINVs \end{pmatrix} + (\text{sequence set } CINVs)}{\text{Index } CINVs \text{ per track}} \right. \tag{20}$$

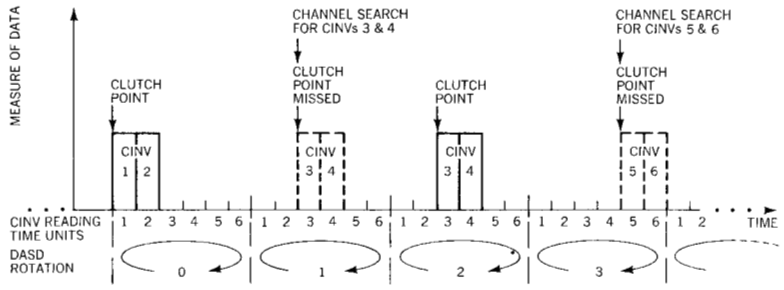
When replicating all of the VSAM KSDS index component, use the following equation:

$$\begin{pmatrix} \text{Tracks for} \\ \text{whole index} \end{pmatrix} \leftarrow \begin{pmatrix} \text{number of} \\ HLI\ CINVs \end{pmatrix} + (\text{sequence set } CINVs) \tag{21}$$

Sequential operations model

We now present techniques for developing a model of VSAM single-string sequential operations. The objective of this section is to discuss the analytic formulas that relate I/O device characteristics, VSAM data set parameters, and CPU instruction pro-

Figure 5 VSAM squential reading and clutch point action



cessing time to the single-string processing time per record. First to be examined are the equations that hold only for an uninserted KSDS, and then discover how clutch-point effects (analogous to card feeding) can be accounted for in this case. The next step is to consider the sequential operations for accessing an inserted file with split and overflow *CA*s. This is followed by looking at sequential performance as a function of insertion level.

To time a sequence of VSAM KSDS sequential read operations, it is necessary to consider the VSAM buffer scheduling rules under OS/VS1 and OS/VS2. Take the number of data buffers and subtract 1. (This buffer is used for *CINV* splittings and is not scheduled in sequential reading operations.) If the remaining number of buffers is equal to or exceeds 4, schedule half of this number (rounded up) for each I/O read channel program. Otherwise, schedule all the available buffers. In APL, these considerations can be expressed as follows:

$$SCHB \rightarrow \lceil (DBF - 1) \div 1 + DOUB \leftarrow (DBF - 1) \geq 4 \qquad (22)$$

The rule for determining the number of buffers to schedule in one I/O channel program (*SCHB*) applies to sequential updating as well. In Equation 22, *DBF* denotes the user-specified number of data buffers. *DOUB* is a logical variable that specifies whether the buffers are scheduled in two sets or one. *SCHB* is also the number of free buffers required before a sequential read operation can be started.

clutch point analogy

The following discussion has an analogy with card readers in which the clutch rotates constantly. The maximum rate of card demand is the rotational rate of the clutch. A demand rate just slightly less than the clutch rotation rate halves the card transport rate. As the card demand rate decreases, each time it becomes less than a unit rotation time, the card transport rate is reduced.

The clutch point effect in DASD devices is illustrated in Figure 5. At the start of the 0th rotation, a clutch point occurs for *CINV*s 1 and 2 together because buffers have been scheduled two at a time ($SCHB = 2$). These two *CINV*s are read in from the six *CINV*s per track ($CIPTRK = 6$). The channel search for *CINV*s 3 and 4 must begin before the clutch point for 3. For illustrative purposes, Figure 5 shows a mismatch between the channel search and the DASD. Since the channel search is shown beginning at (or after) the next clutch point in sequence, the next regular clutch point for *CINV*s 3 and 4 has been missed. That is, *CINV* 3 has partially passed under the DASD reading head by the time the channel search has completed execution. Therefore, reading cannot take place until the next *CINV* 3 and 4 clutch point, which occurs on DASD rotation 2.

Again a channel search (this time for *CINV*s 5 and 6) prevents the reading of *CINV*s 5 and 6 on the next clutch point—units 5 and 6 on rotation 3. *CINV*s 5 and 6 must wait to be read on rotation 4, which is only partially shown. Thus, in this example, five DASD rotations and five clutch points are required to read three units of two buffers each ($DBF = 3$).

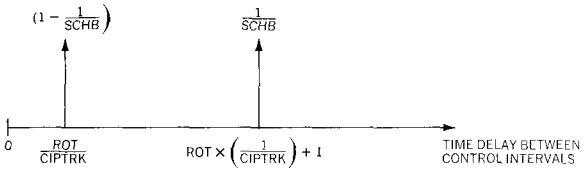
Instead of adding up the number of revolutions for the pattern to repeat itself, define a random variable that accounts for the transition time between *CINV* reads. Figure 6 shows the values and probabilities attached to each value for the case of $DOUB = 0$.

This analysis can be checked by setting $SCHB = 2$ and $CIPTRK = 6$ (assuming that the CPU time is small enough), and then computing $ROT \times [\frac{1}{2} \times \frac{1}{6} + \frac{1}{2} \times \frac{7}{6}] = \frac{2}{3} ROT$ as the average of T_u, which is the average time to read the control interval. This agrees with hand timing results.

Figure 6 shows that, for the values assigned to the random variable T_u, the $(ROT/CIPTRK)$ corresponds to reading within a scheduled buffer set. The $ROT \times [1 + 1/CIPTRK]$ corresponds to reading the first *CINV* of a newly scheduled set of buffers. The 1 in the second term follows from the assumption that it takes just one revolution to move from reading the second *CINV* to reading the third *CINV*. If the CPU time is not small enough, additional rotations may be involved in this transition. Let *PSR* denote the amount of CPU time from the completion of the reading of one set of scheduled buffers until the start I/O for the next set. A more general expression for the average time to read one *CINV* can be written as follows:

$$T_u = ROT \times \left(\frac{1}{CIPTRK} + \frac{1}{SCHB} \times \lceil PSR \div ROT \right) \tag{24}$$

Figure 6 Random variable pattern for average time T_U to read a control interval



This expression is derived by including in the right-hand term in Figure 5 the effects of multiple device revolutions between the scheduling of new sets of buffers. The right-hand term in Equation 24 is the ceiling function of the ratio $PSR \div ROT$, and represents a whole number of revolutions required between scheduled sets of buffers.

Sequential operations with control area splitting

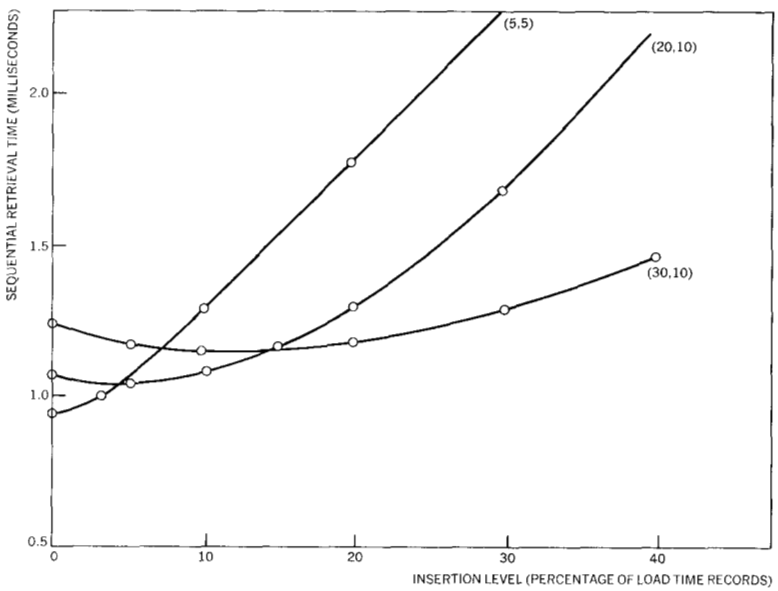
This section indicates how to generalize so as to account for the effects of *CINV* and *CA* splitting on sequential operations. Recall from Figure 1 that, for split *CINV*s, the key sequence order does not correspond to the physical order on a DASD track. In addition, a split or overflowed *CINV* can have a reduced number of records just after the splitting. To account for these effects, we develop a methodology in this section that allows for a mixture of *CA* and *CINV* types. The average time to GET a record sequentially from a KSDS is computed as the following ratio:

$$TSR = \frac{P_u N_u T_u + P_s N_s T_s + P_{ov} Y_{ov} T_{ov} + CCSR}{RECORDS} \qquad (25)$$

Here, the terms in the numerator are defined for unsplit (u), split (s), and overflow (ov) *CA*s. P_u is the probability or proportion of unsplit *CA*s in the KSDS. N_u is the number of *CINV*s in an unsplit *CA*, and T_u is the average time to read a *CINV* in unsplit *CA*. All terms depend on the percentage of insertions made into the KSDS. Similar definitions apply to the remaining terms for split and overflow *CA*s. The term *CCSR* accounts for the *CA*-to-*CA* overhead. *RECORDS* is the average number of records in a control *CA*, and can be computed as an average over unsplit, split, and overflow *CA*s.

Rather than derive the equation for each term, the behavior of Equation 25 is discussed in a qualitative way as the number of additions to the KSDS grows. At low insertion levels, *RECORDS* increases in proportion to the level of additions. If there is sufficient free space, N_u or T_u does not increase, and P_s and P_{ov} re-

Figure 7 Sequential retrieval time versus insertion level for several choices of free space parameters



main zero. Under these circumstances, *TSR* decreases. As the level of additions grows, P_s and P_{ov} become significant. It follows that *RECORDS* decreases, since *CINV* splittings tend to halve the number of records contained within a *CINV*. At large insertion levels, the terms T_u, T_s, and T_{ov} grow, since these *CA*s have many *CINV*s out of physical sequential order.

Figure 7 shows how *TSR* depends on the insertion level for several values of the free space parameters *FCI* and *FCA*. Notice that both the (20, 10) and (30, 10) designs show a decrease in *TSR* at a five percent insertion level. The (30, 10) design has poorer sequential performance at the zero insertion level, since the added free space carries no records. The (20, 10) design shows a balance of good zero-insertion performance and a moderate increase in *TSR* up to the fifteen percent insertion level.

direct addition time versus addition level

As a final illustration of the techniques used to understand VSAM performance, we now analyze the time to make a direct addition, using the computation of *PCIS* in Equation 2 and *PCAS* in Equation 9. Recall that a direct addition results in three kinds of response from VSAM. If free space is available, the addition looks very similar to a direct update. If there is inadequate free space in a target *CINV*, then a split *CINV* occurs. If, in this last case, no free *CINV* is available, a *CA* splitting occurs.

Suppose $I = INS \times REC$ additions are made, and the average insertion time for all these additions is to be computed. Let B denote the fraction of the additions that cause at least a $CINV$ splitting. From Equations 14 and 11, there are $CIPCA$ load-time $CINV$s per CA and $\lceil REC \div RECPCA$ load-time CAs. Define the parameter B as follows:

$$B \leftarrow PCIS \times CIPCA \times (\lceil REC \div RECPCA) \div (REC \times INS) \quad (26)$$

since $PCIS \times CICPCA$ is the average number of split $CINV$s. The computation that results in Equation 26 includes those $CINV$-splitting additions that result in CA splittings. Let C denote the fraction of CA-splitting additions as follows:

$$C \leftarrow PCAS \times (\lceil REC \div RECPCA) \div (REC \times INS)$$

The average insertion time can be written as an average of three times as follows:

$$AIT \leftarrow (TSCA \times C) + (TSCI \times (B - C)) + (1 - B) \times TD_u \quad (27)$$

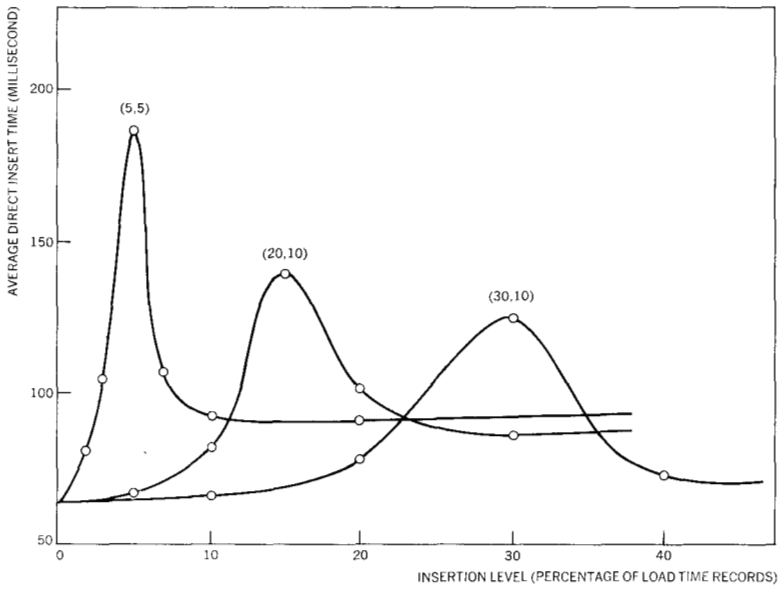
The term $TSCA$ is the time for a CA splitting; $TSCI$ is the time for a $CINV$ splitting; and TD_u is the time for a direct updating. Formulas are not derived for these terms, but the relative size of each term is noted. $TSCA$ is large, i.e, of the order of seconds. $TSCI$ is of the order of hundreds of milliseconds; and TD_u is usually less than a hundred milliseconds.

It might be expected that AIT increases as B and C increase with insertion level. Since each CA splitting creates free space, after many CA splittings, the effect of $TSCA$ and $TSCI$ is expected to lessen. AIT is the average over all additions up to the current insertion level. We can derive the average value of an insertion in a small band of insertions from Equation 27. The value obtained in this way depends on how small a band is taken. Figure 8 shows the derived AIT value for three designs: (5, 5), (20, 10), and (30, 10). Note the increase in AIT due to CA splitting. After these CA splittings occur, further additions find free space in the $CINV$s, and the AIT decreases to a value moderately higher than the zero insertion value. This happens because the extent of the KSDS has expanded and unsplit $CINV$s continue to split even after the data set has doubled in size.

Concluding remarks

Our intention has been to offer an appreciation of VSAM performance sensitivities and to relate those sensitivities to data set design parameters. This has been done in the hope that designers may become more aware of the effects of their choices among those parameters. This discussion has been restricted to VSAM as it applies to IBM operating systems. Effects of paging,

Figure 8 Average insertion time versus insertion level for three choices of free space parameters



the VSAM catalogue, and VSAM multirequest strings—which may be important in some cases—have not been included.

Two main areas for attaining data set performance objectives using VSAM have been presented. In direct operations, VSAM performance tends not to degrade as new records are added to a file. Also, sequential operations in an inserted file tend to perform as well as in an uninserted file. In both cases, infrequent reorganization of the data set is required.

Although VSAM offers gains in performance, these gains depend on the proper planning of the data set layout. There is thus a need for analysis of the expected data set usage and need for mapping that usage into the design parameters available.

APPENDIX

To prove that all P_i have the same probability distribution we introduce a new set of random variables $\{Q_i, i = 1, 2, \cdots, REC\}$, which have a symmetric distribution, and are simply related to the $\{P_i, i = 1, 2, \cdots REC/NRCI\}$.

$$\begin{aligned} Q_1 &= F_K(X_1^*) \\ Q_2 &= F_K(X_2^*) - F_K(X_1^*) \\ &\vdots \\ Q_{REC} &= F_K(X_{REC}^*) - F_K(X_{REC-1}^*) \end{aligned} \tag{28}$$

The $\{P_i\}$ are related to the $\{Q_i\}$ by the following equations:

$$P_1 = Q_1 + Q_2 + \cdots + Q_{NRCI}$$
$$P_2 = Q_{NRCI+1} + \cdots + Q_{2NRCI} \tag{29}$$
$$\vdots$$
$$P_{REC/NRCI} = Q_{1+(i-1)NRCI} + \cdots + Q_{iNRCI}$$

Equation 29 is understood by noting that the $F_K(X_\alpha{}^*)$ cancel in any sum except for the leading and trailing terms. If we show that the $\{Q_\beta\}$ are symmetrically distributed, then it follows that the $\{P_i\}$ are identically distributed. This last step is taken by explicitly computing the density of the Q from the known density of the $F_K(X_2{}^*)$.

Feller[3] gives probability density of the $F_K(X_\alpha{}^*)$ as follows. Simplify notation by defining

$$Z_k = F_K(X_k{}^*) \quad k = 1, 2, \cdots, REC$$

where

$$0 \leq Z_1 \leq Z_2 \leq \cdots \leq Z_{REC} \leq 1$$

The density for $\{Z_1, Z_2, \cdots, Z_{REC}\}$ is given by

$$f(z_1, z_2, \cdots, z_{REC}) = REC!$$

for

$$0 \leq z_1 \leq z_2 \leq \cdots \leq z_{REC}$$

Now invert the relationship given in Equation 28. The inverse transformation is shown in Equation 30 as follows:

$$Z_1 = Q_1$$
$$Z_2 = Q_1 + Q_2$$
$$Z_{REC} = Q_1 + Q_2 + \cdots + Q_{REC}$$

The Jacobian of the transformation in Equations 30 is clearly 1. Hence the density for the $\{Q_a\}$ has the following simple form:

$$f_Q(q_1, q_2, \cdots, q_{REC}) = R!$$

for

$$0 \leq q_i$$

where

$$i = 1, 2, \cdots, REC$$

and

$$q_1 + q_2 + \cdots + q_{REC} = 1$$

Since this density is uniform in all its variables, it follows from Equation 29 that all $\{P_i\}$ have the same density.

ACKNOWLEDGMENT

The authors wish to acknowledge the support of Harry Hill and Stephen Goldstein in carrying out the work reported here.

CITED REFERENCES

1. *OS/VS Virtual Storage Access Method (VSAM) Planning Guide*, Form No. GC26-3799, IBM Corporation, Data Processing Division, White Plains, New York 10504.
2. R. E. Wagner, "Indexing design considerations," *IBM Systems Journal*, **12**, 4, 351–367 (1973).
3. W. Feller, *An Introduction to Probability Theory and its Applications*, Volume II, John Wiley and Sons, New York, New York (1971).