PERFORMANCE ASPECTS OF IMS

by

ROBERT GERARD MEAGHER

B.Sc., St. Francis Xavier University, 1978

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

June, 1980

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

June 25, 1980

Abstract

This paper deals with the tuning of the data base management system IMS. We will look at IMS in some detail; in particular, we will consider those aspects of IMS which are thought to have a measurable effect on performance at an IMS installation. These are: the original design of the data base, the adoption of logical relationships and secondary indexes and the choice of access method. Other features are thought to have a lesser effect but some of these will also be considered. These aspects will be evaluated by their ability to decrease processing time, increase flexibility and/or decrease storage requirements.

The proposals presented in the first part of the paper will be further evaluated in the final chapter. The experiences of three IMS users will be incorporated to support or refute these proposals, as well as to provide insight into the kinds of performance studies being done, when they are done and what results are expected.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to sincerely thank Dr. Paul Gilmore for his guidance, constructive criticism and financial support for this project. I am indebted to Gary Watson of B.C. Hydro, Salim Nuraney of I.C.B.C. and Emile Côté of B.C. Tel for their valuable donation of time and experience.

Chapter 1 - Prelude

Data Base Management Systems

As we enter the 80's the data and information processing requirements of modern society are becoming increasingly complex and vast. As methods of production and marketing have progressed in leaps and bounds, so too has the data processing industry. As recently as 20 years ago, data processing departments (as we know them today) were nonexistent. Today they are the very heart of the organization.

As the size and complexity of organizations have increased, their information needs have increased even more rapidly. No longer is it possible for one individual to understand completely all aspects of the organization. Hierarchies of control develop. Jobs become more and more specialized. Organizations are affected not only by local demands and tastes but also by governmental legislation and world trade. Facts and figures must be collected, stored and made available for future reference - for the day to day operation of the company, to satisfy legal requirements and for management decision making.

> "The importance of data to the functioning of the organization, coupled with the large investment in developing and maintaining the data base, emphasizes that data should be managed to the same degree as other valuable corporate resources."[1]

A data base is a collection of these facts and figures

for use by the organization. Although not necessarily, the data base is often stored on tape or disk with a computer being used for storage, retrieval and processing. It is often the case that these data bases contain great quantities of data of various kinds which are required for a variety of purposes by several users. Such a scenario has prompted the development of a particular piece of software known as a Data Base Management System (DBMS).

Basically, a DBMS provides a means by which the data can be shared by different users to reduce redundancy and inconsistency and to enforce standards and security restrictions. It is, in fact, a tool for managing the data.

Many commercial data base management systems have been developed and marketed. The generalized features of these systems make them adaptable to the particular requirements of different organizations, and once adopted, modifications are possible to reflect the changing data processing requirements of the enterprise.

Data base management systems have been on the market for the past 10 years but only recently have they been used extensively. New developments in data base technology and applications are making it increasingly evident that a DBMS is a requirement in most data processing environments.

"At the 1979 Industry Briefing held by International Data Corporation, it was projected that data management software of all types will account for over half of the proprietary software market revenues in 1983 (estimated to be $2.3 billion). A sizeable portion of this amount will be spent specifically for data base management systems."[2]

Information Management System

IBM's Information Management System (IMS) is one of the oldest data base management systems on the market and yet it is one of the most sophisticated in terms of options and features available. It is also, by far, the most widely used of all DBMSs. In 1979 the total number of users of IMS was approximately 1200.[3]

IMS has gone through several stages of development. The initial implementations ran under the operating system OS/360 and were known as IMS/360 Version 1 and IMS/360 Version 2. The current version is IMS/VS Version 1 which runs under OS/VS (Operating System/Virtual Storage). This paper will be concerned with the latter which for the sake of brevity will be shortened to IMS/VS or simply IMS.

Performance

With growing information processing demands it becomes increasingly important that this processing is executed within an acceptable time frame and for an acceptable cost. Thus, the concept of performance, and in this case the performance of a DBMS, must be considered.

> "... performance may be interpreted as the technical equivalent of the economic notion of **value**. That is, performance is what makes a system valuable to its users. Like value, performance is only one of the two faces of reality in the economic world, the other side being **cost**."[4]

To achieve a specified level of performance for all users, the data base designer must be concerned with the cost/value tradeoffs associated with each potential use of the data base and must choose those designs which optimize the overall performance. Such performance evaluation studies are an ongoing concern and do not yield a permanent solution. As an enterprise's data processing requirements change, it will be necessary to consider these changes (as for the initial design) from a performance standpoint.

With a data base management system at least two aspects of performance are evident - those which concern the developers of the DBMS and those which concern the individual installation and its particular application. This latter aspect, which can best be described as tuning, will be the subject of this paper. Because of the nature of a DBMS, its performance is very much dependent upon the environment in which it is being run and in particular, the data processing requirements and the nature of the application data of the installation.

Synopsis

This paper is concerned with data base management systems, in particular, IMS/VS. It is also concerned with performance, in particular, the tuning of IMS/VS. What will be presented in the ensuing chapters will be of importance to system designers and data base administrators both during the initial design of the IMS application and whenever any major modifications are made at the installation.

Only the most important aspects - those which affect performance most - will be considered. One could easily go on at great length outlining minute details and their probable effect upon performance, however, if there is no noticable improvement in performance, the study is futile from a practical standpoint, and that is what it is about - practicality.

In Chapter 2, IMS/VS will be discussed in sufficient detail sc as to briefly introduce the reader to the subject. It will then be possible to consider performance in context. Chapter 3 deals with the design of the IMS data base - how the data base should be organized; Chapter 4 covers the access methods used - how the data bases are arranged on secondary storage; and Chapter 5 treats three other considerations - buffering, application programming and main storage requirements. Chapter 6 will related the tuning experiences of three IMS installations to the aspects of performance introduced in Chapters 3, 4 and 5.

Chapter 2 - IMS Overview

Introduction

In this chapter an attempt will be made to introduce the basic structural entities of IMS. This treatise will in no way be complete. It is offered here as background material to the ensuing chapters and hence will be oriented towards those things which affect the performance of an IMS installation.

For an indepth coverage of this topic, the interested reader is directed to the following sources: Chapters 13-18 of Date[6], pages 84-168 of the IBM Systems Journal [27], the IMS/VS General Information Manual [16] and the IMS/VS Primer [19]. All of the above are very readable and thorough. For information not contained in these, one should consult the other IMS/VS manuals listed in the bibliography.

Data Base Organization

IMS/VS is a hierarchical data base management system. A particular installation will probably consist of several data bases such as the one illustrated in Figure 2.1.

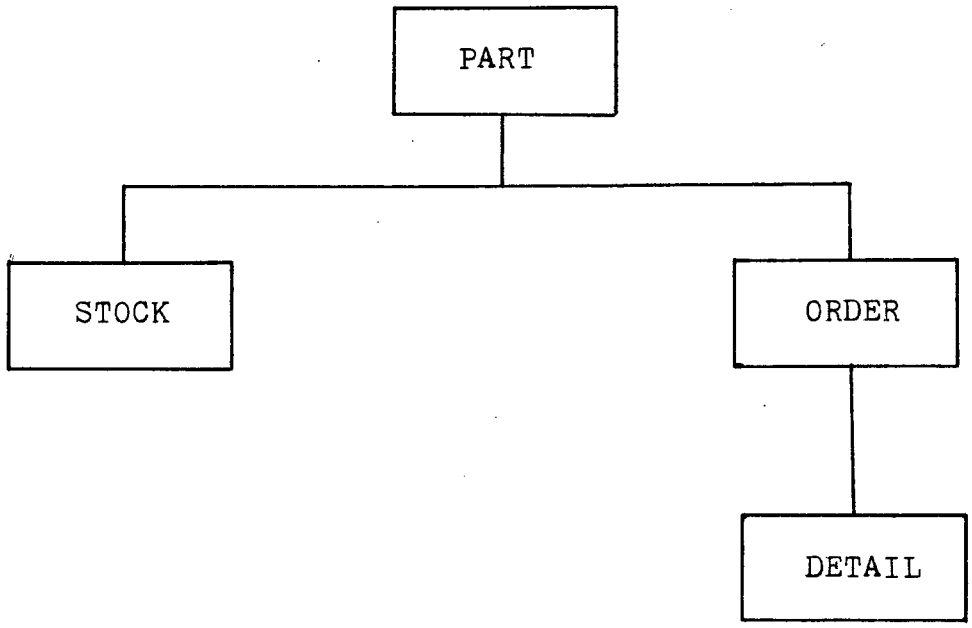

Fig. 2.1 Logical data structure of the PARTS data base.

This is a logical representation of the physical data base. Logical refers to the way in which the data base is viewed by various users. Physical refers to the way in which the data is actually stored on magnetic tape or direct access storage devices.

A hierarchy is made up of parent-child relations. In this example PART is the parent of both STOCK and ORDER because it is immediately above it in the hierarchy. In a similar manner, ORDER is the parent of DETAIL. Correspondingly, DETAIL is a child of ORDER and both STOCK and ORDER are children of PART. All children (which are of the same type) of a given parent are known as twins. For example, all occurrences of STOCK for a given PART are twins.

PART is said to be the root of the hierarchy because it

has no parent. Each root occurrence determines one data base record which may contain several occurrences of STOCK, ORDER and DETAIL. There will of course be several occurrences of PART and thus several data base records. PART, STOCK, ORDER and DETAIL are said to be segments. The segment is the smallest amount of data which may be transferred by one access to the data base. However, a segment is made up of one or more fields, for example, the PART segment may contain fields which represent the part number, the part name and the part color.

One of the fields of a segment may be designated as the segment sequence field or key. If a segment has a key, segments of that type occur under their parent segment in ascending or descending key value sequence. For root segments, the key must be unique. Other segments may have multiple occurrences of the same key, but when inserting such a segment it is necessary to specify if it should be placed FIRST, LAST or HERE (current position in the hierarchy) with respect to other occurrences of the same key.

The sequence field is also used as all or part of a symbolic pointer to a segment in a data base. The symbolic pointer is actually the concatenation of the keys in the sequence fields of all segments that must be retrieved to reach the desired segment, including the sequence field key of the desired segment.

All segments that fall below another segment (i.e., its children, its children's children, etc.) are known as its

dependents. The hierarchical sequence is the order in which the hierarchy is stored. In IMS the hierarchical sequence follows a preorder traversal of the tree (see [2], pp. 84). When the data base of Figure 2.1 is stored, it will consist of the first occurrence of PART followed by all STOCKs associated with that PART followed by all ORDERs associated with that PART. Each ORDER will be followed immediately by all DETAILs associated with that ORDER. All of this comprises one data base record and will be followed by the next occurrence of PART and thus the next data base record.

Data Base Design

When designing a data base for a particular installation, there are many features of IMS which may be adapted to best suit the environment. Three such things will be outlined here, namely: logical relationships, secondary indexes and access methods.

Logical Relationships

As was noted, each IMS location will probably contain several data bases. It is often the case that the same data must be carried in several data bases. This redundant data can result in inconsistencies and increase the work required in insertion, deletion and updating. To prevent this, logical relationships should be used.

Logical relationships provide a facility to interrelate

segments from different hierarchies (data bases). In doing so, new hierarchical structures are defined which provide additional access capabilities to the segments involved. These segments need not come from different data bases, though, as it is possible to logically relate segments from the same data base.

The basic mechanism used to build a logical relationship is to specify a child segment as a logical child, by relating it to a second parent, the logical parent. Figure 2.2 illustrates such a logical relationship.

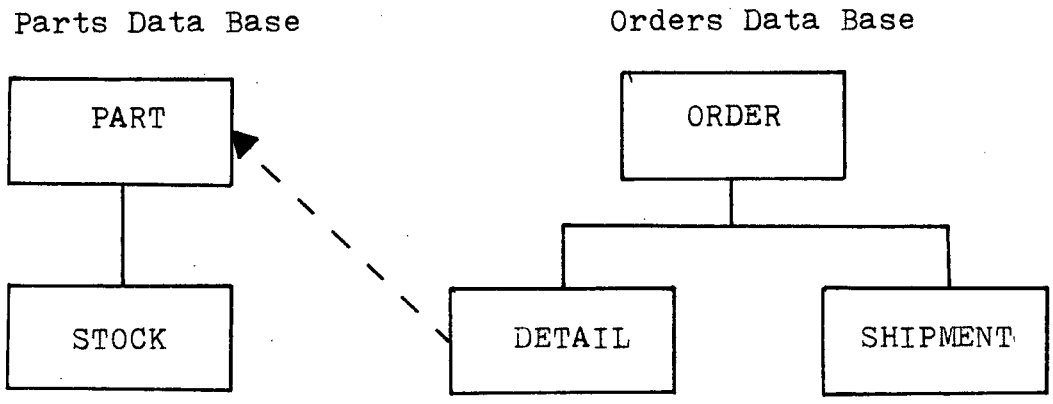

Fig. 2.2 Logical relationship between Parts and Orders data bases.

In this illustration, DETAIL and SHIPMENT are both physical children of ORDER. By making DETAIL a logical child of PART, PART becomes the logical parent of DETAIL. Thus, two access paths are provided to DETAIL, through PART and through ORDER.

Of course, this structure is transparent to the user when he queries the data base. Special circumstances arise, though, when an attempt is made to insert, delete or replace. For example, what if a new DETAIL segment is to be inserted and the corresponding PART does not exist? It is necessary to explicitly state the course of action to be followed if such a situation arises.

Secondary Indexes

In IMS, a secondary index can be used to index a given segment on the basis of any field of that segment or on the basis of any field in a dependent of that segment. The field on which the index is based may be a concatenation of up to five such fields (from the same segment) taken in any order.

Each secondary index represents a different access path to the data base other than via the root key. Such an access path can provide faster retrieval of data. Figure 2.3 shows an example of a secondary index.

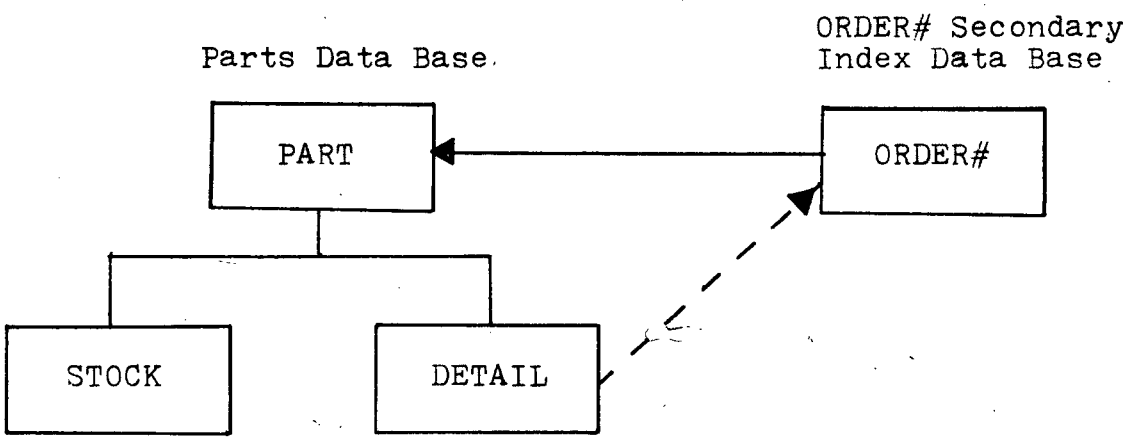

Fig. 2.3 Secondary indexing.

It can be seen from this illustration that a new data base is established which is known as an index data base. In this example, the segment ORDER is the index source segment as it contains the field ORDER# on which the index is constructed. Each segment in the index data base contains an ORDER# field and a pointer to the appropriate segment in the Parts data base. These segments are known as index pointer segments and are ordered on the index field. The segment pointed to by the index data base is known as the index target segment.

Of course, the index source segment and the index target segment may be the same segment. The index target segment is usually, but not always, the root segment of the data base.

Access Methods

IMS has two access levels for the storage of data base records. The first level consists of the routines within the IMS control program. These routines process the second level access methods which reflect the organization of the data on tape or disk. See Table 2.1 and Figure 2.4.

Processing Routines:
Hierarchical Sequential Access Method(HSAM)
Hierarchical Indexed Sequential Access Method (HISAM)*
Hierarchical Direct Access Method (HDAM)
Hierarchical Indexed Direct Access Method (HIDAM)
Physical Record Interfaces:
OS/VS Indexed Sequential Access Method (ISAM)
OS/VS Sequential Access Method (SAM)
OS/VS Virtual Storage Access Method (VSAM)
IMS/VS Overflow Sequential Access Method (OSAM)

*HISAM can also be used as a physical record interface.

Table 2.1 Processing Routines and Physical Record Interfaces for use with IMS data bases.

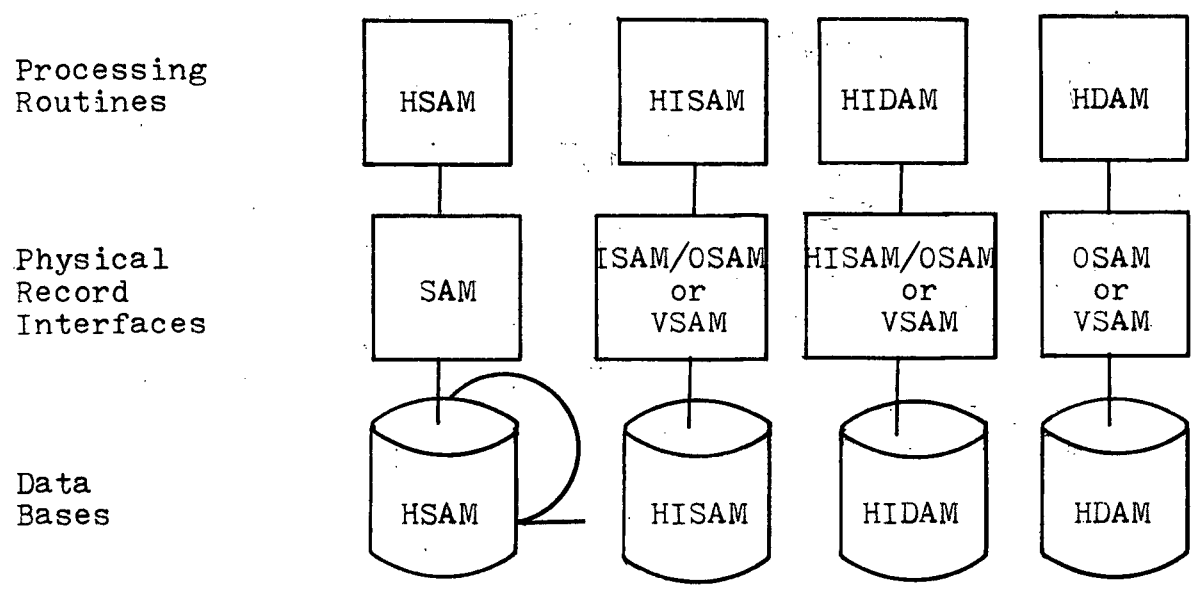

Fig. 2.4 Access methods and storage structures.

The actual data is stored in data sets, the organization of which is determined by the physical record interface. For example, an HSAM data base is stored in a SAM data set. In the case of a HISAM or a HIDAM data base, two data sets are needed (one for the index and one for the rest of the data). If VSAM is chosen as the physical record interface, the index data set is known as a Key Sequenced Data Set (KSDS) and the other data set is kncwn as an Entry Sequenced Data Set (ESDS). Otherwise, ISAM or HISAM and OSAM are used for these purposes.

In an HSAM data base, the hierarchical sequence is represented entirely by physical contiguity. The segments of each data base record are stored in hierarchic (preorder) sequence, in one or more consecutive data set records. The last segment of one record is followed immediately by the root segment of the next data base record. See Figure 2.5 for an illustration.

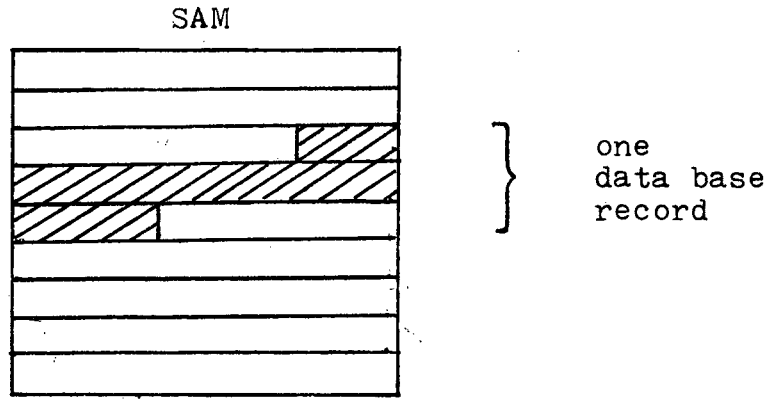

Fig. 2.5 HSAM implementation.

HISAM allows for indexed access to the root segment and

sequential access to dependent segments. It requires two data sets for implementation. These may be ISAM and OSAM or if VSAM is used, these two are replaced by KSDS and ESDS respectively.

Again, data base records are stored in physically contiguous locations in hierarchical sequence. However, the storage area is divided into an indexed area (ISAM or KSDS) and an overflow area (OSAM or ESDS). The root segment and as many segments of the data base record as can be accommodated are stored in the former, the latter is used to store the remainder of the record. Physical pointers are used to chain the index area part of the record to the part of the record stored in the overflow area and to chain subsequent data set records in the overflow area, if more than one is required. See Figure 2.6 for an illustration.

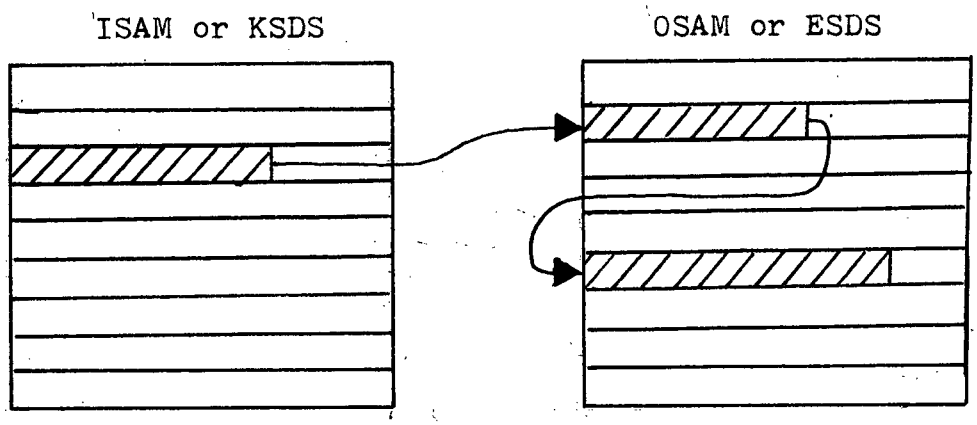

Fig. 2.6 HISAM implementation.

HDAM provides direct access to the root segment by a hashing function and chaining. The storage area is made up of

a series of data sets which are divided into a root addressable area and an overflow area. Either OSAM or ESDS are used for these purposes. Each data set holds all occurrences of a given set of segment types. The record structure is maintained by pointers which connect the segments in hierarchical sequence.

The root addressable area is used to store root segments and a limited number of dependent segments. Synonym chains are used to connect all root segments that hash to the same data set record. Dependent segments of a data base record are stored in the overflow area and are connected by physical pointers. See Figure 2.7.

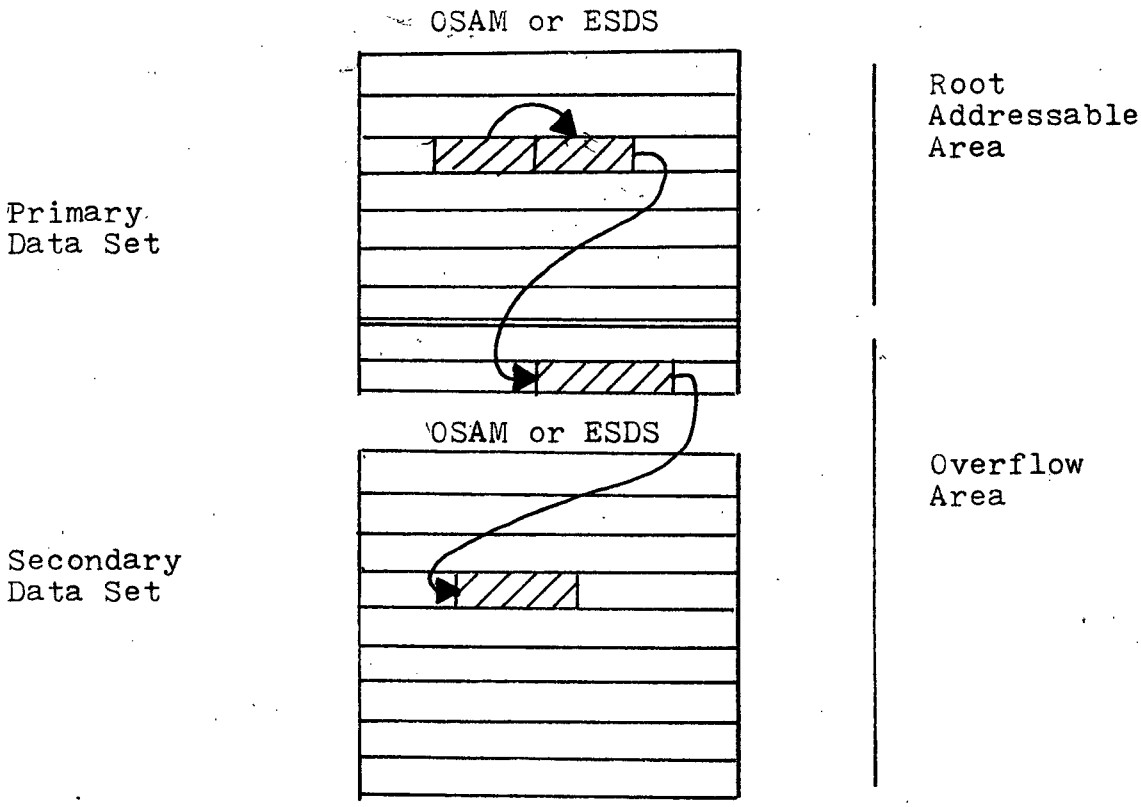

Fig. 2.7 HDAM implementation.

HIDAM is similar to HDAM except that record sequencing is maintained through an index data base which is implemented as an ISAM or KSDS data set. See Figure 2.8.

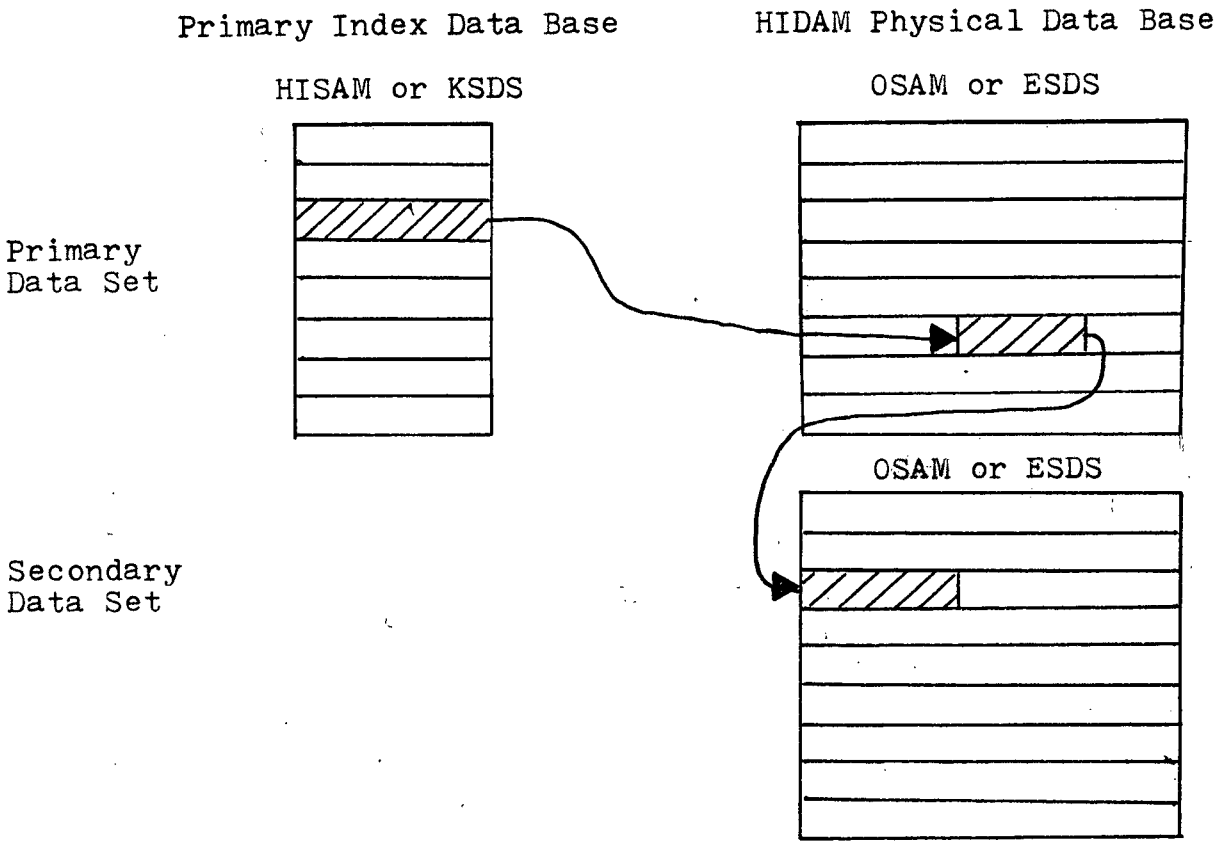

Fig 2.8 HIDAM implementation.

Control

IMS/VS is a control program that provides data management and data communication management services. It operates under the control and with the facilities of OS/VS as one of the OS/VS processing programs. It therefore requires one or more OS/VS regions which, once created, are referred to as IMS/VS

regions. Figure 2.9 shows the overall system flow.

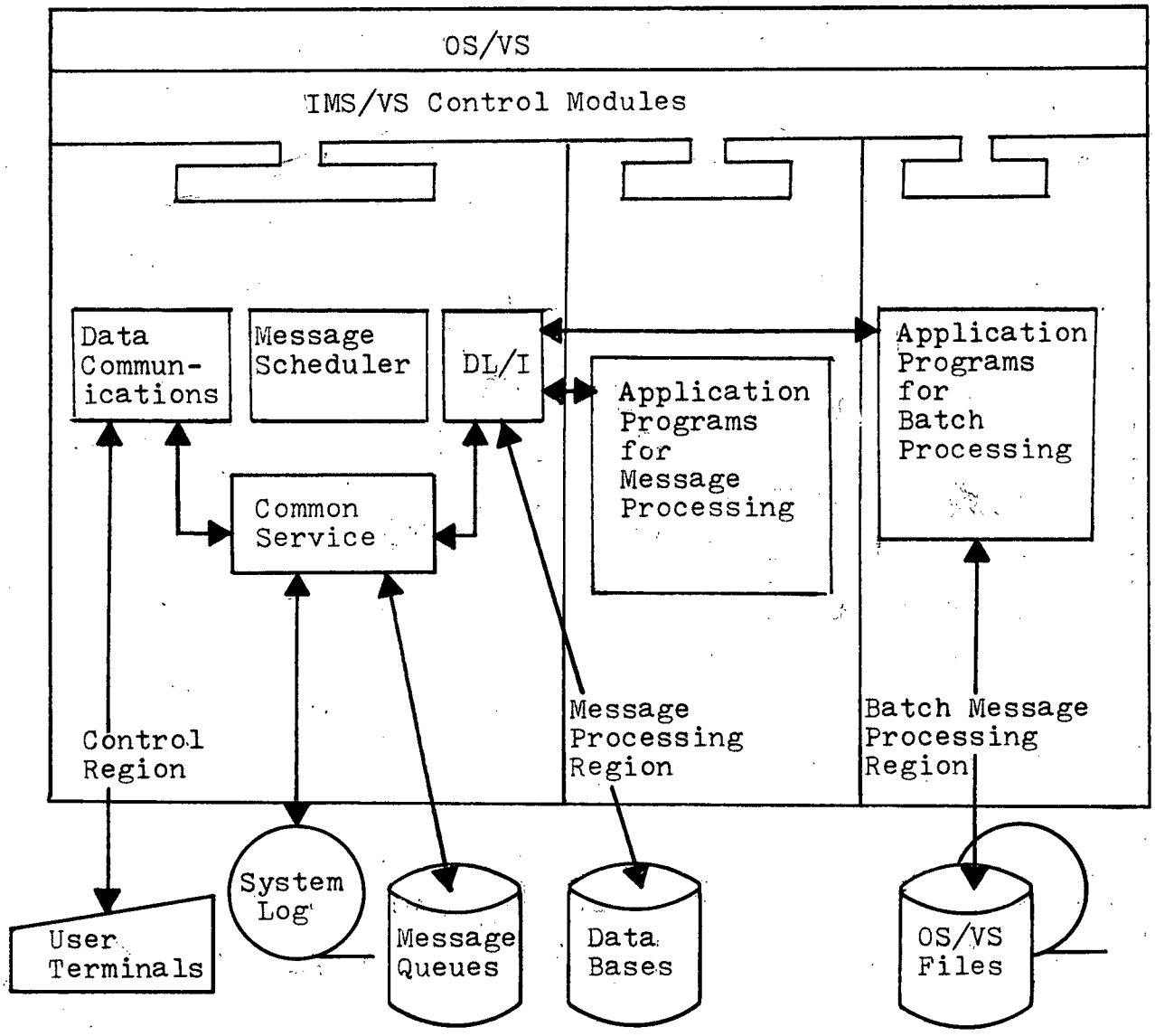

Fig. 2.9 IMS/VS system flow.[5]

The OS/VS nucleus and its resident extensions provide the service modules and access methods necessary for an IMS/VS application program.

The data communication feature of IMS/VS provides

facilities that permit users to communicate through terminals with a single on-line execution of the system. Communication may take the form of: (1) transmission of messages between terminals and user-written application programs; (2) transmission of messages between terminals; and (3) transmission of commands and command responses between terminals and the on-line execution.

When an input message is received from a terminal, the data communication facility calls the common service and the message is logged and queued. Once messages are queued and waiting for processing and a region is available for message processing, control is passed to message scheduling. There it is determined from the message prefix which message processing program is needed. If necessary, this application program is loaded; control is then passed to it.

The application program will then make requests for input messages and/or data from the data bases. These requests are handled by Data Language/I (DL/I) which references either common service (for queued messages) or IMS data bases (for data base requests). The application program may also generate its own messages for queuing or make modifications to the data bases.

These application programs are written in a host language (PL/I, COBOL or Assembler Language) from which DL/I may be invoked by subroutine calls. There are nine such DL/I subroutines. See Table 2.2 for a list and descriptions.

Code	Description
GU	Get Unique
GN	Get Next
GNP	Get Next Within Parent
GHU	Get Hold Unique
GHN	Get Hold Next
GHNP	Get Hold Next Within Parent
ISRT	Insert
DLET	Delete
REPL	Replace

Table 2.2 DL/I subroutine codes and descriptions.

These DL/I calls enable the application program to navigate through the data base. A position pointer (as many as one for each possible path in the hierarchical tree) marks the progress through the data base in a preorder traversal of the data base tree.

When the application program terminates or requests another input message, all its queued output messages are transmitted to the designated output terminals.

IMS/VS can also be used for batch processing. In this case, a subset of the IMS/VS control facility, the batch application program and DL/I all operate out of a single OS/VS region. These batch processing jobs are submitted as ordinary

OS/VS jobs and cannot use the IMS/VS data bases concurrently with the message processing programs.

For each application program (batch or message processing) a program specification block (PSB) is created. It defines the application data structure required by that application program. Each PSB contains one or more program communication block (PCB), one for each logical data base the program intends to use. The PCB provides a mapping from the logical data base to the physical data base. It also specifies the kinds of accesses allowed by the program, ie. read only, update, insert, and/or delete.

Telecommunications

The telecommunications facility of IMS/VS provides the linkage between terminals and the remainder of the IMS/VS system. Because of the large number of terminal types supported by IMS/VS, the concept of a logical terminal has been provided. A logical terminal is an abstraction of a real terminal in which only those aspects necessary for writing application programs and the operation of the system are apparent.

One physical terminal can have one or more logical terminals associated with it. The IMS/VS user refers to a logical terminal and never has to worry about physical terminal addresses. If a physical terminal is rendered

inoperative, the logical terminals normally associated with that terminal can be assigned to another physical terminal.

The master terminal is a logical terminal that acts as the operational centre of IMS/VS. It has complete control over IMS/VS and is responsible for such things as initial start-up, monitoring and for dynamically altering the operation of the system.

All input and output messages are written to a system log to assist in backout and recovery. Each message is also queued in main storage with direct access storage for backup as required. In this way, messages can be received by the system although the resources necessary to process them may not be immediately available.

A message formatting service (MFS) allows application programs to deal with logical messages instead of device dependent data. MFS uses information about the format of messages going to and coming from devices and application programs to do the formatting. This allows the user to change the presentation of data on a device without actually changing the application program.

Utilities

IMS/VS provides many utilities to aid in the day to day running of the system. Three such utilities will be introduced here, namely: checkpoint/restart, reorganization

and statistical analysis.

Checkpoint/Restart

To ensure the integrity of its data bases and message processing, IMS/VS uses checkpoint/restart. At regular intervals during IMS/VS execution, checkpoints are written to the log tape (see Figure 2.9). This is to limit the amount of reprocessing required in the case of an emergency restart. A checkpoint is taken after a specified number of log records are written to the log tape or after a checkpoint command is issued by an application program.

In case of failure, IMS/VS is restarted using the current log tape. The data base changes made by incomplete application programs are backed out and output messages generated by these programs are deleted. The input messages already processed are then requeued. Message processing programs are then restarted and batch jobs are resubmitted.

Reorganization

Over time, it may become necessary to reorganize one or more of the data bases. This is expedient when a number of inserts and deletes cause segments of a data base record to be no longer physically adjacent or when deleted segments continue to occupy storage space. Some structural changes may be made during reorganization, for instance, a secondary index or a logical relationship may be added.

There are three major steps in reorganization: (1) unload

the data base; (2) delete the old space, redefine the new space and optionally make structural changes; and (3) restore the data base. Each of these steps involves the running of several utility programs dependent upon the type of storage structure used and the number and kind of structural changes to be made.

Statistical Analysis

The IMS/VS statistical analysis utilities provide statistical information about online IMS/VS operation. The information is collected from the IMS/VS system log. Various reports are available giving such things as the number of messages queued but not sent, message traffic for lines, physical terminals and logical terminals and total and average CPU task times. This information is useful in determining system load characteristics and for detecting bottlenecks.

Chapter 3 - Data Base Design

Introduction

The design of an IMS data base is neither unique nor straight forward. Depending on the installation, the design stage can last for several months to a year and, being a repeating process, the analysis-design-implementation cycle will probably exist for the entire life of the application.

In the past, data base design has been regarded primarily as a process of conversion - adapting the existing files and data structures to make them compatible with the DBMS. A more open-minded and definitive approach is needed.

> "Data base generation should not be regarded as a conversion problem, but as an opportunity to plan the organization, use and management of data. The emphasis should be on analysing the data requirements of a business or other enterprise, and on the accurate reflection of these requirements in the schemas."[6]

Fortunately, the trial and error approach to data base design is being replaced by a more purposive methodology.

> "Data base design has moved a long way from being a black art toward being a rigorous science, but there is still room for the artist. The designer must still discover from time to time a way of doing something that seems impossible."[7]

In this chapter the performance components of data base design will be considered. The characteristics of a data model which permit the necessary processing within an

organization will be outlined. Following this, the design of the data base record will be examined and finally the performance aspects of logical relationships and secondary indexes will be looked at.

First some definitions. Entities are concepts or objects of interest to an organization, something about which facts are gathered and kept.[8] For example, departments and employees may be thought of as entites. An entity may be connected to another entity by a relationship; this defines a mapping between the two entities. For example, departments are made up of employees; the relationship going from departments to employees is that of employment. Entities also have attributes. An attribute is a property of an entity and would not normally be meaningful except when considered with the entity to which it belongs. For example; name, salary, wage and marital status might all be considered attributes of an employee. A transaction is the basic building block of an enterprise's data processing needs. It is an action or set of actions which require access to the data base. For example, an employee is added to a department or an employee's salary changes. See Table 3.1 for further examples.

Data Model Characteristics

The choice of IMS as a data base management system imposes severe constraints on the way in which the data base

content may be viewed. The hierarchical model contains inherent characteristics which may or may not be present in a relational or network model. Because of these characteristics, the appropriateness of the hierarchical model is dependent upon the data processing requirements of the organization.

In general, IMS (or any hierarchic DBMS) is most expedient when the structure to be modelled is hierarchical in nature. This occurs when the mapping between entities is one-to-many. For example, each department has many employees and each shipping order is made up of several parts each of which may have several components. Pure hierarchies are hard to find though - an employee may actually work in two different departments and what is considered a component for one order may actually be a part for another order. These many-to-many relationships, while not impossible to model, usually entail duplication of stored data or extra processing when a hierarchical structure is imposed upon them.

Before work can be done on the design of the data base an extensive analysis of the organization must be made in order to determine the data requirements. In this analysis, entities, relationships, attributes and transactions will be isolated. For each, pertinent information such as probability frequencies, volumes, security constraints and priority of access will be collected or projected so that they can be used in the design stage.

Armed with this information the system designer will set

about modelling the organization with IMS data bases. Entities will be grouped into one or more data bases. Each data base will reflect relationships between entities and must be constructed so that the various transactions can be performed as easily and as cheaply as possible without severely affecting the performance of the system.

The data dictionary is an important tool which can be used to assimilate this information and provide the connection between the analysis and design stages. IBM has developed a program product known as the Data Base/Data Communication Data Dictionary (DB/DC Data Dictionary) which supports IMS/VS. When the DB/DC Data Dictionary is adopted the user has a computer processable dictionary in which he can store and retrieve definitions, data descriptions, relationships of data and program information. The dictionary deals with entities and the attributes which characterize entities and relationships. For more information see the DB/DC Data Dictionary General Information Manual[7].

The Data Base Record

The organization of data into data base records can be the definitive design choice with regard to the system's performance. A good data base design should: (1) reflect the organization of the data in the real world, (2) permit the processing of as many different transactions as necessary with

a minimal amount of effort, and (3) meet performance standards so as not to over-tax the total system. These are often conflicting objectives and the choice of a compromise design may be a difficult task.

For each data base it is first necessary to select a root. All other entities in the data base must be related (either directly or through other entities) to the root in a parent-child type relationship.

> "Unfortunately, hierarchical methods are incapable of supporting more than a single root; consequently, they have forced a generation of designers to select a single entity type as supreme when in fact there are usually several that are logical peers, that is, neither superior nor inferior, neither independent nor dependent, neither parent nor child."[9]

Because of the way IMS data base records are stored, the root is always the first segment of the record to be accessed, making it faster to retrieve the root than any other segment. The choice of a root will also determine the number of data base records - one for each root occurrence - as well as the order in which the records will be accessed; except for HDAM, records are stored in ascending or descending order based on the value of the root key.

For illustrative purposes, a data base which is to consist of the entities, attributes and relationships given in Table 3.1 will be considered. Starting from these initial definitions we will proceed to develop an IMS data base. The reader should note that the resulting design will not be optimal, rather, it provides an initial data base upon which

improvements can be made depending on the users own performance demands. Neither will the resulting design (nor intermediate designs, for that matter) be unique. Even in the simple example given there may be choices other than the ones chosen.

Entities:	Attributes:
ORDERS	Number
PARTS	Number, Color, Size
CUSTOMERS	Number, Address, City, Routing

Relationships:

Each order has an associated part or parts.

Each order has an associated customer.

Each city has an associated routing.

Table 3.1 Entities, attributes and relationships of the ORDER data base.

Since this data base is to process transactions concerning the filling and shipping of orders, we will call it the ORDER data base. Since both PARTS and CUSTOMERS are related to ORDERS we will select ORDERS to be the root of the data base, leaving the attributes of PARTS and CUSTOMERS to be dependents of ORDERS. Figure 3.1 shows the hierarchical

representation of such a data base so far.

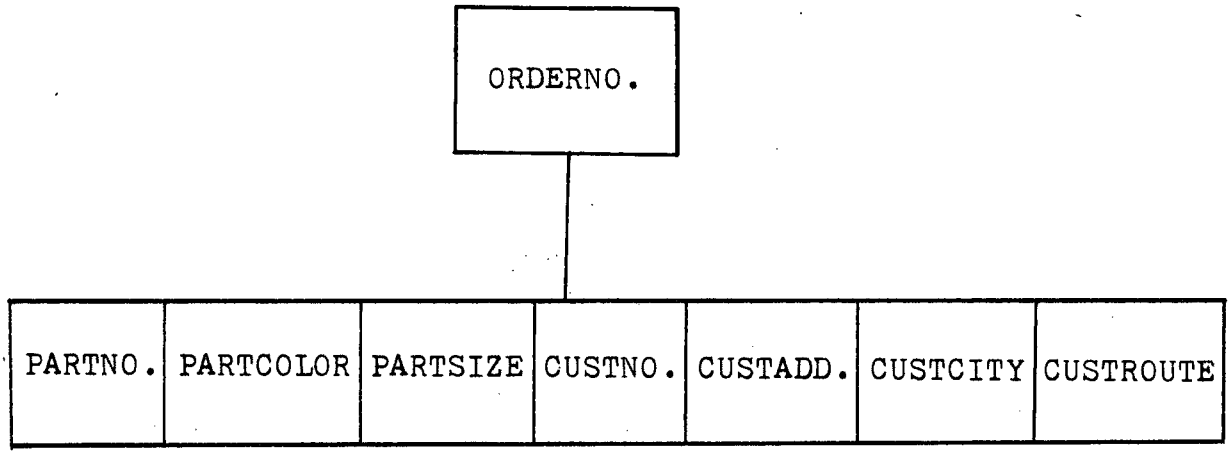

Fig. 3.1 ORDER data base - first normal form.

We will use normalization (see Date [6] and Gearhart [12]) to produce a viable IMS hierarchy. Figure 3.1 is said to be in first normal form, it consists of a root and all entities (and their attributes) which are related to the root.

The second normal form can be achieved by distinguishing subordinate keys for the various levels in the hierarchy. These keys, when compounded with the root of the data base become the concatenated keys of the segments in the IMS data base. Associated with each key will be a set of attributes. Together they form an IMS segment. Figure 3.2 shows the ORDER data base in second normal form. PARTS and CUSTOMERS have been broken into two separate segments, the keys of which are PARTNO. and CUSTNO.

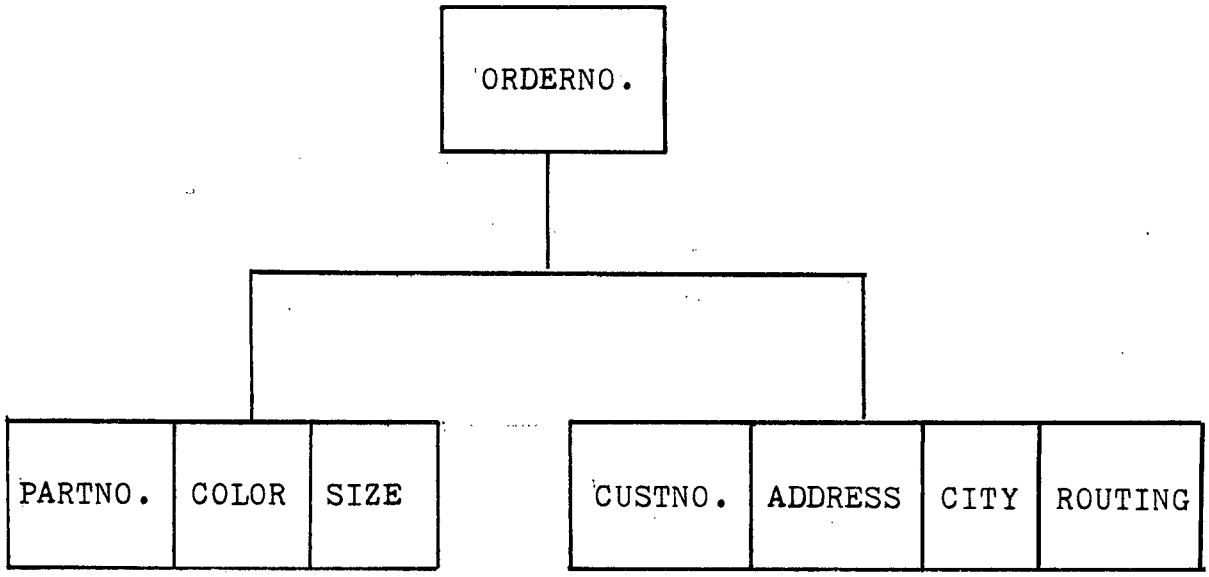

Fig. 3.2 ORDER data base - second normal form.

At this point it may turn out that some attributes of a segment are actually related to other attributes in the segment. In this case, third normal form can be obtained by removing these attributes from the segment, making one of them the key of a new segment and making the new segment a child of the old segment. In the case of the ORDER data base, ROUTING was actually related to city and not CUSTNO., therefore a new segment was formed with CITY as a key and ROUTING as an attribute. Figure 3.3 shows the data base in third normal form.

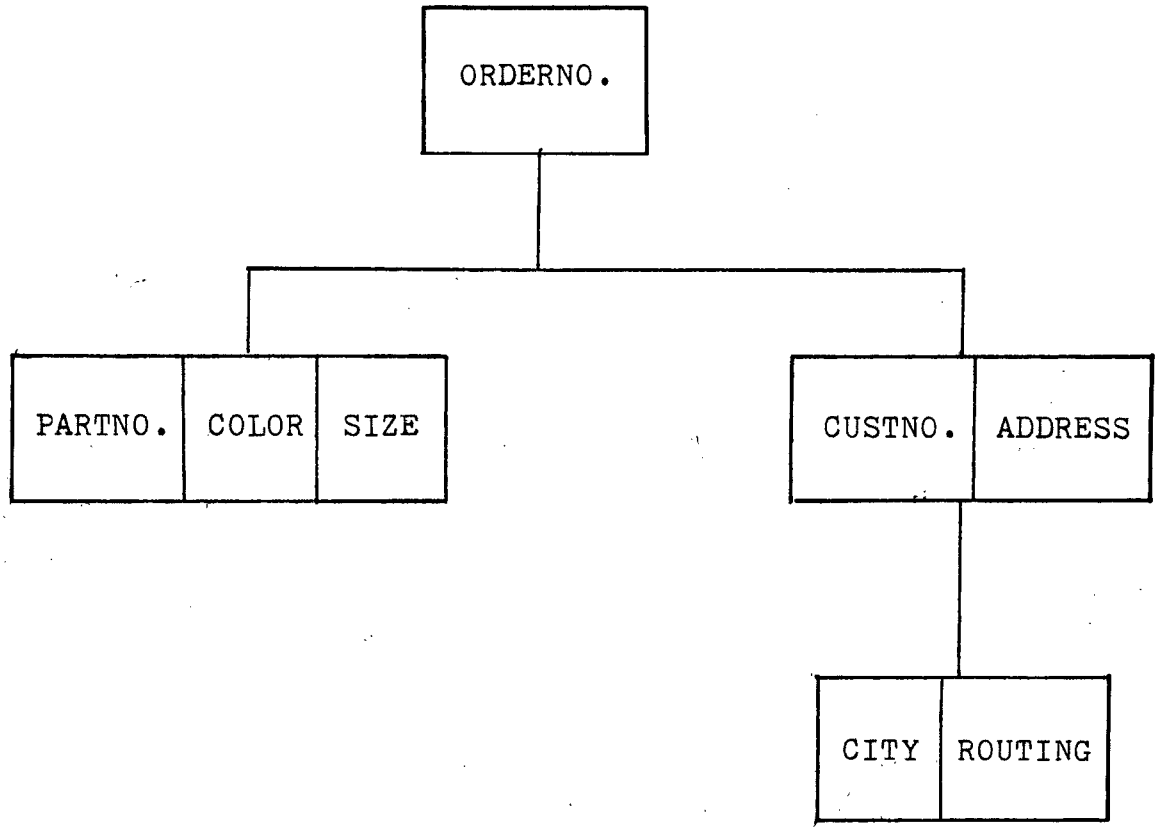

Fig. 3.3 ORDER data base - third normal form.

If a segment has two or more attributes each of which may take on more than one value for a given key value, further normalization is necessary. Fourth normal form requires that only one of these attributes be kept in the segment and that the others be removed to a dependent segment. In our example both COLOR and SIZE are multivalued; each part comes in a number of colors and in a number of sizes. Figure 3.4 shows the data base in fourth normal form where SIZE is removed to a dependent segment.

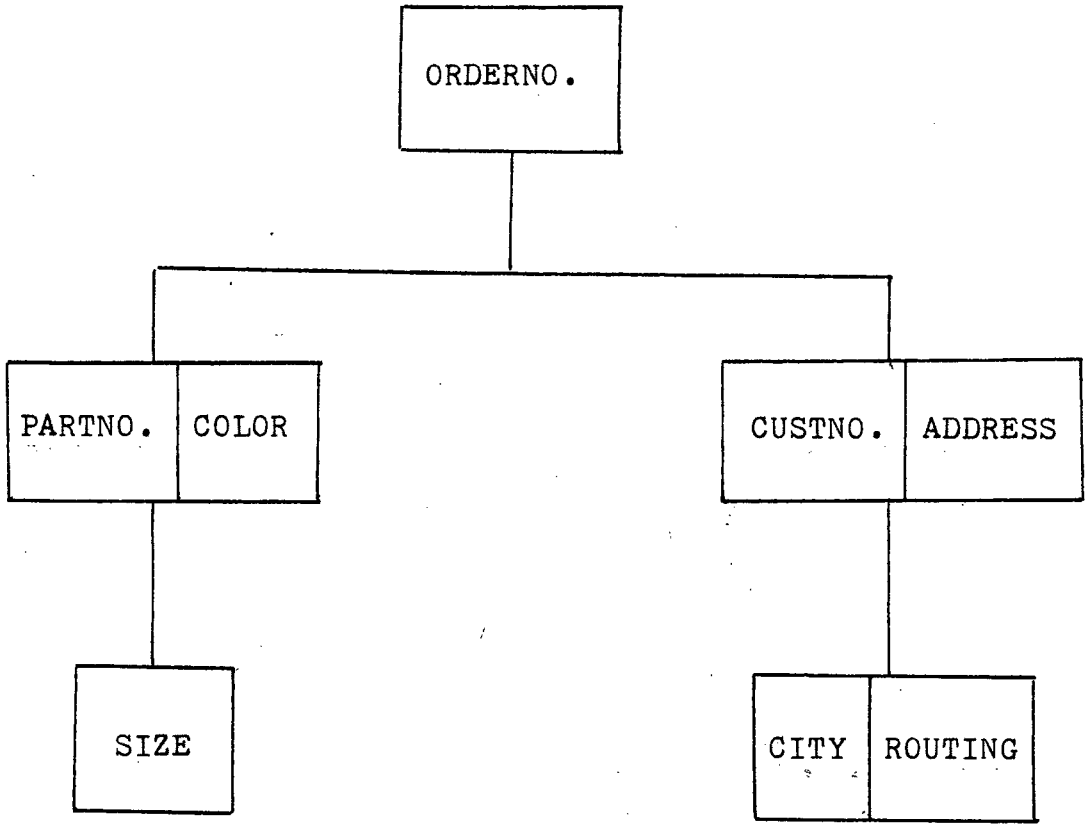

Fig. 3.4 ORDER data base - fourth normal form.

Such a process of normalization will produce a working data base, but not necessarily one which is optimal from a performance standpoint. Several factors must be considered which will affect the storage requirements and/or access times of the data base. These factors may dictate that new segments be created, that two or more segments be consolidated into one or that subtrees be repositioned.

Table 3.2 summarizes the general guidelines to be followed to determine whether fields should be grouped into

one segment or should be left in separate segments.

Group in one segment ◄——► Separate segments	
Few occurrences (<3)	Multiple occurrences (>10)
Small (<20 bytes)	Large (>100 bytes)
High use (every access to record)	Low use (once a month)
Read-only	Update, insert, delete
General use	Secured use

Table 3.2 Grouping attributes into segments.[10]

Very large and very small segments can have a negative effect on the data base. Long segments will require larger block or control interval sizes leading to wasted space when smaller segments are put in the same block or control interval. Performance is improved when all segments of the data base are about the same size.

When a segment is exceptionally long an improvement in performance may be sought in one of two ways. If the long segment is seldom used it may be desirable to place it in a separate data set group (see Constraints and Extensions, Chapter 4). If the long segment has a number of fields associated with it, it may be possible to break the segment into two segments based upon the usage of the fields. However, adding a new segment does increase the processing

necessary to access the dependent segment and should only be undertaken when the advantages are clear.

If a segment has a variable length field, the field should be placed at the end of the segment. Fields that do not always occur in a segment should be made into variable length fields so that space is conserved when the field does not occur. It should be remembered, though, that variable length fields increase the amount of processing necessary especially if a number of inserts and/or deletes are expected.

When developing a final design, it must be remembered that segments within a data base record are stored in hierarchical order. That is, segments are stored from top to bottom within the hierarchy and from left to right within subtrees. Therefore, segments that are referenced most frequently should be close to the top left of the hierarchy. The least frequently accessed segments should be in the bottom right of the hierarchy.

It is important to note that to access a segment it is necessary to access all of the segments on the hierarchical path up to that point, unless a secondary index is used. These implied accesses should also be considered when the ordering of the tree is being determined.

Logical Relationships

As pointed out earlier, duplication of data not only

wastes storage space but also increases the overhead in maintaining two copies of the data. Logical relationships reduce these problems by enabling one copy of the data to be accessed from two different data bases. These connections between or within data bases may be either unidirectional or bidirectional, and in the case of bidirectional, may be implemented by physical pairing or virtual pairing.

Logical relationships utilize physical or symbolic pointers to establish the relationships. Physical pointers are of the type used to implement physical data bases and consist of a relative byte address of a segment from the beginning of a data set. Physical pointers can only be used with HDAM or HIDAM and not with HISAM data bases. Symbolic pointers are the concatenation of the keys in the sequence fields of all segments that must be retrieved to reach the desired segment.

A unidirectional logical relationship allows the user to traverse between the data bases in one direction only. It is implemented by placing a logical parent pointer (either physical or symbolic) in one of the segments. Figure 3.5 gives an example.

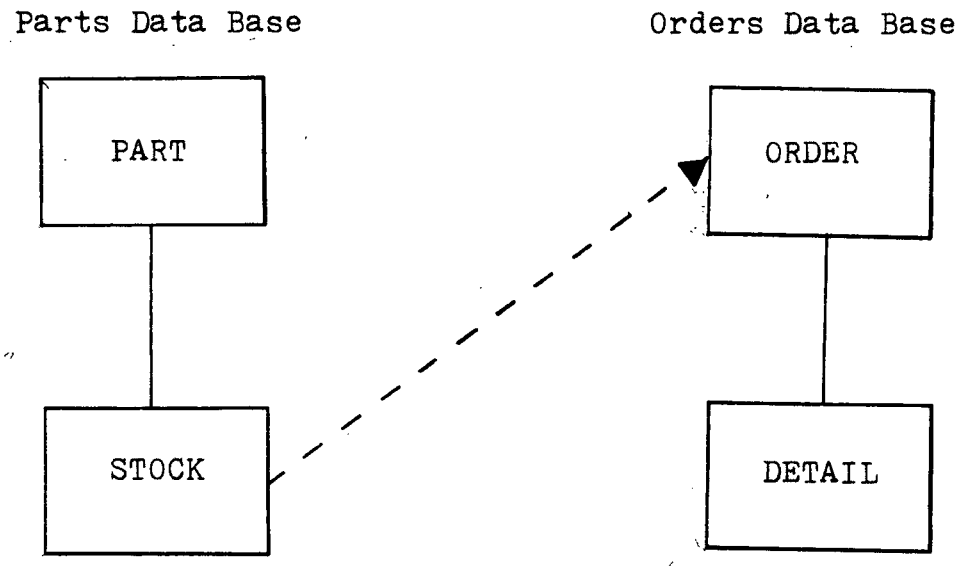

Fig. 3.5 Unidirectional logical relationship.

In this case, each STOCK segment will contain a pointer to its logical parent ORDER.

In a bidirectional logical relationship using physical pairing a dependent segment is repeated and a logical parent pointer is placed in each of the two duplicate segments. Two paths are created each in the same way as the single unidirectional path was created. See Figure 3.6.

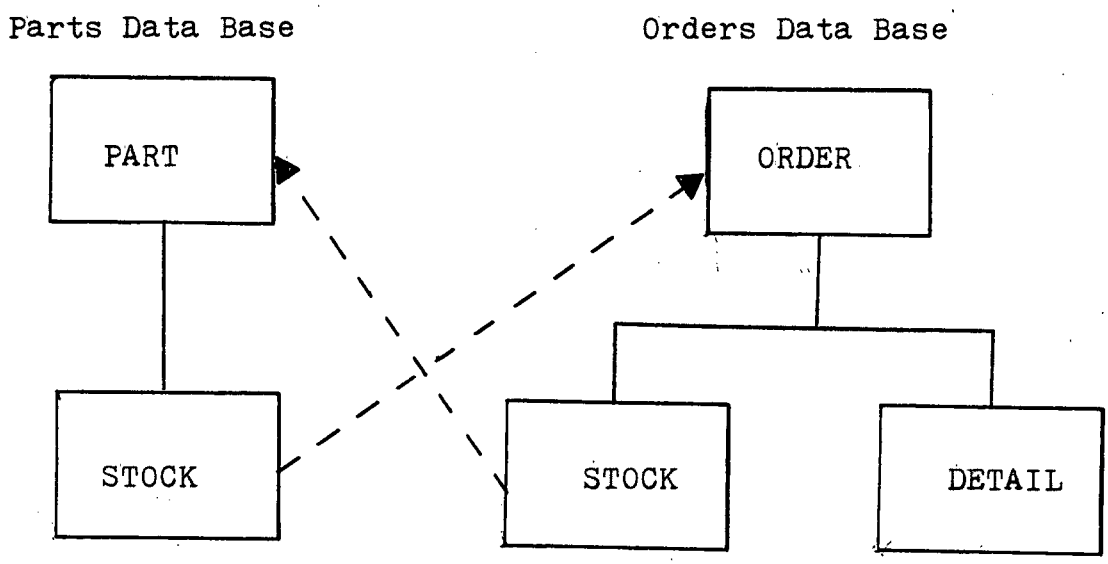

Fig. 3.6 Bidirectional logical relationship using physical pairing.

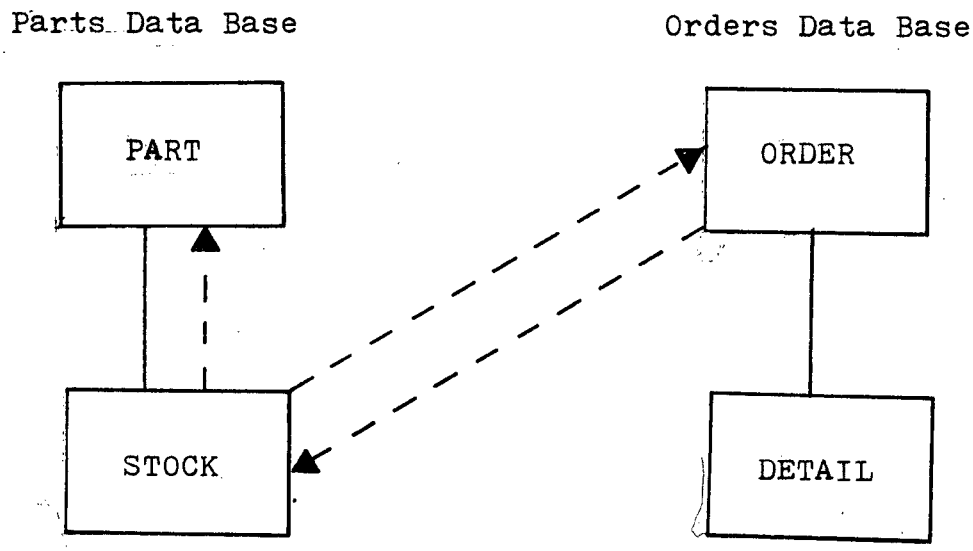

Fig. 3.7 Bidirectional logical relationship using virtual pairing.

In a bidirectional logical relationship using virtual pairing, the path in one direction is implemented in the same way as for the unidirectional case. The reverse path, in this case, is implemented by chaining ORDER and all associated STOCKS with logical child and logical twin pointers and placing a physical parent pointer in each STOCK occurrence. See Figure 3.7.

Regardless of whether physical or virtual pairing is used, similar results are reached. Paths are provided from PART through STOCK to ORDER and form ORDER through STOCK to PART. In the unidirectional case only one of these paths can exist. The choice of unidirectional or bidirectional logical relationships is dependent upon the processing requirements. The user should realize that more processing and storage overhead is required when bidirectional logical relationships are used and so unidirectional relations should be chosen over bidirectional whenever possible. For a more detailed treatment of logical relationships see Chapter 4 of the Systems/Applications Design Guide[20].

The choice of physical or virtual pairing can have a significant impact on performance. Physical pairing is best suited to the application in which accesses are made from both sides of the relations. In our example, this would mean that it is equally probable to want to go from PART to ORDER as to go from ORDER to PART. It must be remembered, though, that the STOCK segment is repeated and this requires overhead in storage as well as processing (maintenance of the second copy

is the responsibility of the system and not of the user).

This duplicate segment is not required in virtual pairing. However, virtual pairing heavily favors one path over the other. In our example, the path from PART to ORDER is equivalent to that in the physical pairing example. To go from ORDER to PART, however, requires that a twin chain be followed. Each retrieval of a segment along this chain must be considered as an I/O and therefore may require a great deal more processing (depending on the length of the chain) than the other path. Because of this, virtual pairing favors the applications in which accessing is mostly from one side (or in which the twin chain is assured to be short) and in which the number of updates of the logical child segment is greater than for physical pairing.

A choice must also be made between physical and symbolic pointers. Physical pointers provide fast access to the logical parent, however, symbolic pointers tell more about the segment being considered. When physical pointers are used, the segment must be accessed before its concatenated key can be determined. If the logical parent's concatenated key (or a part of it) is used as a search argument, time can be saved by using a symbolic pointer as it is not necessary to go into the other data base to determine the key. A symbolic pointer requires more storage space but this space is easily justified if searches such as this are common.

When symbolic pointers are used exclusively, time is saved in reorganizing the data base as it is not necessary to

gather, sort, match and apply the addresses used with the direct pointers.

When virtual pairing is selected it is possible to select twin forward, twin backward, logical child first or logical child last pointers or a combination of these. Logical child last is useful when segments are to be inserted or deleted at the end of the chain and logical child first when segments are to be inserted or deleted at the first of the chain. Twin backward pointers are useful when the segment to be accessed is likely to be near the end of the chain and twin forward pointers when the segment to be accessed is likely to be near the beginning of the chain. Combinations of these have the effect of the combined advantages, however, it should be noted that each pointer requires additional storage space and additional overhead for maintenance.

Secondary Indexes

Secondary indexes can be used to establish an alternate access path to a data base. They are particularely useful in data bases with a large number of segment types when it is necessary tc provide direct access to one of the dependent segment types. Processing time can be saved by going directly to the desired segment as opposed to tracing the hierarchical path. They can also be used to establish a processing order other than that in which the data base is stored.

When secondary indexing is used an index data base is created. This data base is implemented in a VSAM data set and contains a search field and a direct address or symbolic pointer to the target segment. If HISAM is used the pointer must be symbolic. Direct address or symbolic pointers can be used for HDAM or HIDAM data sets. Secondary indexes are not allowed in HSAM data sets. The index may be created selectively or sparsely by suppressing the creation of an index entry for some of the source segments.

The source field used to define the secondary index need not be unique. If a non-unique source field is used, subsequent records with the same source field are stored in a secondary data set and chained to the primary data set. In order to produce unique keys (and save having to access a secondary data set), a subsequence field may be specified. This field will also be taken from the source segment and is stored in the index data base as well.

The index data base itself may be used as a small extract data base. By duplicating a frequently used field the user can have quick access to this data and the maintenance of the data base is provided by the system (i.e., changes in the larger data base are reflected in the index data base automatically).

The records within the index data base are stored in ascending key value (source field) sequence. This provides for an alternate processing sequence. If the target field is the root of the data base, processing of this data base may be

performed in an order other than that in which it is stored. This is particularely useful when several application programs wish to use the same data base in a number of different ways.

When a secondary index has been defined, access is provided (through the index) to the index target segment type and all segment types in the hierarchic path of the index target segment. This includes the parents as well as the dependents of the target segment. Figures 3.8 and 3.9 illustrate this point. Figure 3.8 shows a data base for which a secondary index is to be defined.

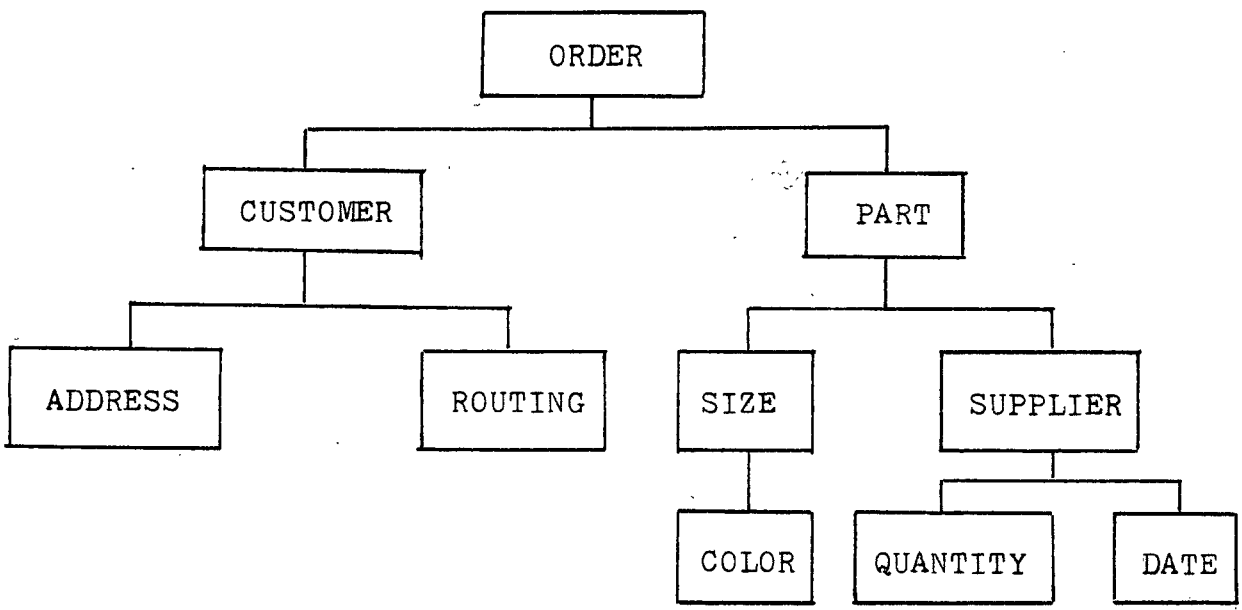

Fig. 3.8 An order processing data base.

Let us say that the SUPPLIER segment is both the source and the target segment of a secondary index. Then a secondary data structure will be defined as shown in Figure 3.9.

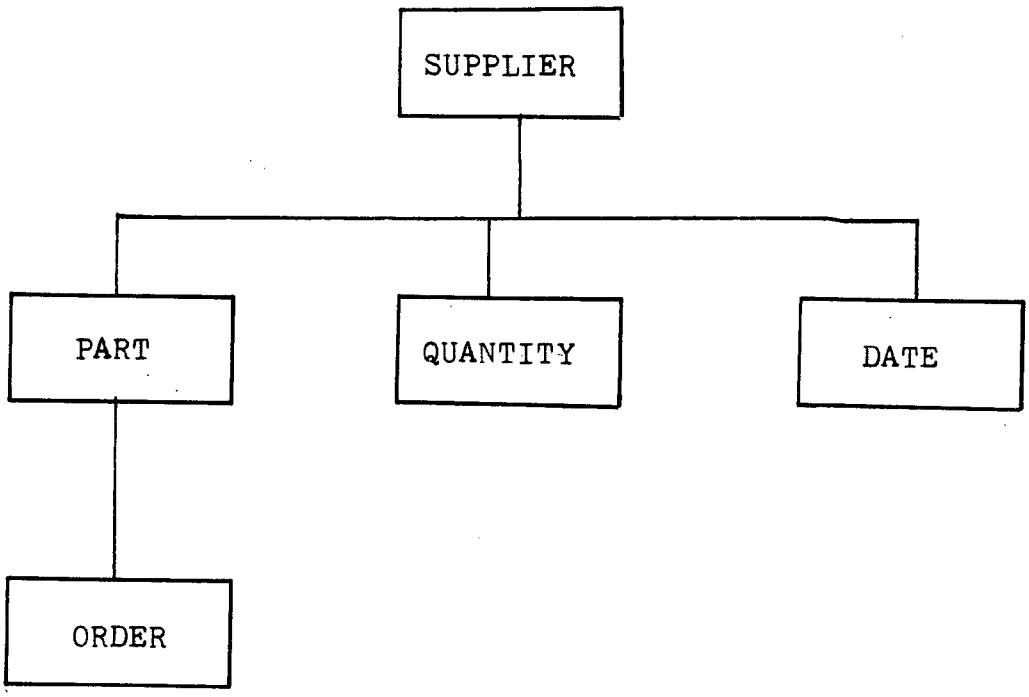

Fig. 3.9 Secondary data structure.

The secondary data structure is formed by making the index target segment type the root. The parent segment types of the index target segment type are then placed in reverse order in the left-hand subtree. The other dependents of the target segment type are then placed in the right-hand subtrees.

When considering the adoption of a secondary index the extra processing required to maintain the index data base must be taken into account. An index allows access to a segment

with one I/O in addition to the index I/O. This should be compared with the number of I/Os normally required to access the segment. An additional I/O is required each time an index record is created or destroyed and two I/Os are required for each modification. The time required to create the index data base initially and to reorganize it should also be considered. Such things may be outweighed if the number of changes to the source field is small or if the transactions which will use the secondary index are of high pricrity.

Not to be neglected are the actual storage requirements of the index data base. As was pointed out, each VSAM record contains a duplicate of the source field and a pointer to the target segment. In addition to this there is some control information and (optionally) subsequence fields. This can run into a substantial amount of storage space and for this reason such things as selective indexing or shorter search fields may be considered.

Chapter 4 - Access Methods

Introduction

In this chapter the access methods introduced in Chapter 2 will be dealt with in more detail. In particular, the four processing routines and their corresponding physical record interface (see Fig. 2.4) will be evaluated and compared with respect to the two performance measures - time and space.

The performance of an access method is dependent upon the structure of the file in which the data is stored. The problem remains, therefore, to select an access method which is optimal for the data base that is to be stored and for the queries that are to be applied to the data base. But as pointed out by Yao and Merten [42], this may be a difficult task:

> "No universally optimal file structure exists and no general method is available for selecting an optimal file structure for a particular application."[11]

However, the choice of an access method may be one of the most important factors in determining performance at an IMS installation.

> "Various file organizations seek to perform better in certain respects for certain data base and typical query characteristics."[12]

For these reasons, much of the evaluation contained in

this chapter is relative and serves only to compare the access methods.

In what follows, each access method will first be looked at in greater detail. Attention will be paid to the underlying storage structure and the ideal processing environment for each. Following this each access method will be considered with regard to the time/space tradeoff. Finally, some constraints and extensions will be considered.

HSAM

In an HSAM data base segments are stored in hierarchical sequence and so must be loaded in that order. Data base records are stored in ascending or descending key sequence (if a key is specified) and must be loaded and retrieved in that order.

Each SAM data set consists of a number of fixed length blocks. The records are stored sequentially starting with the first record and proceeding in hierarchical sequence for subordinate segments. Each block will contain several segments. If there is not sufficient room at the end of the block to store a complete segment, the remaining space is left blank and the segment is stored in the next block. When one record ends the next record follows immediately, in the same block if possible. See Figure 2.5.

The only major performance criteria for an HSAM data base

is the selection of an optimal block size. The block must be at least as large as the largest segment and should be selected so as to mimimize the amount of unused space left at the end of the block.

Processing is done sequentially through the data base so that when one segment is found the search for the next begins where the last left off. To process the data base randomly it is necessary to start at the first of the data set for each search and scan sequentially until the desired segment is found.

Updating requires that the entire data base be rewritten onto another tape or disk. For all practical purposes then, insertion, deletion or replacement of segments is not possible.

For these reasons, HSAM data bases are generally only used for data which is not expected to undergo many changes and for which sequential processing (eg. producing reports) is the usual mode of access. HSAM is also useful as backup for other data bases as it requires a minimal amount of storage space and can be kept on tape. Also, its structure lends itself to the dump/reload format of a backup data base.

HISAM

With a HISAM data base either ISAM/OSAM or VSAM may be selected as the physical record interface. In either case the

data base consists of an ISAM or KSDS data set which contains the root and as many of the dependent segments as can fit in one record and an OSAM or ESDS data set which contains the remainder of the dependents and, in the case of ISAM/OSAM, any data base records which are inserted after initialization.

Each data set is made up of logical records. Each logical record contains as many segments (in hierarchical sequence) as will fit, pointers to maintain the hierarchic sequence among logical records and some unused space. See Figure 2.6.

For ISAM/OSAM the logical record is of the form illustrated in Figure 4.1.

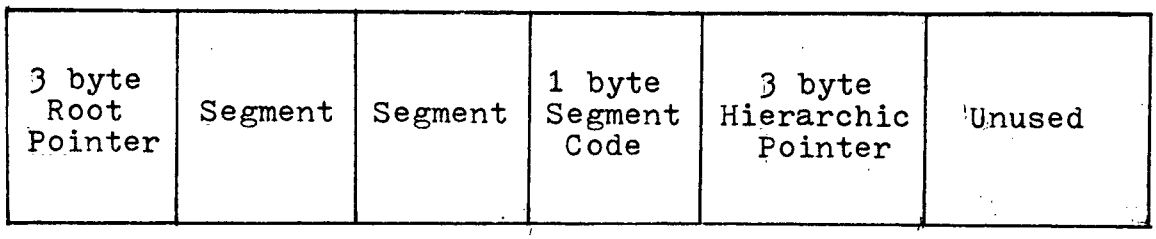

Fig. 4.1 Logical record of an ISAM/OSAM implementation of a HISAM data base.

The three byte root pointer specifies the next root in root-key sequence which has been added since initialization

and is therefore in the OSAM data set. The three byte hierarchic pointer specifies the logical record which contains the next segment in hierarchic sequence.

For VSAM the logical record is of the form illustrated in Figure 4.2.

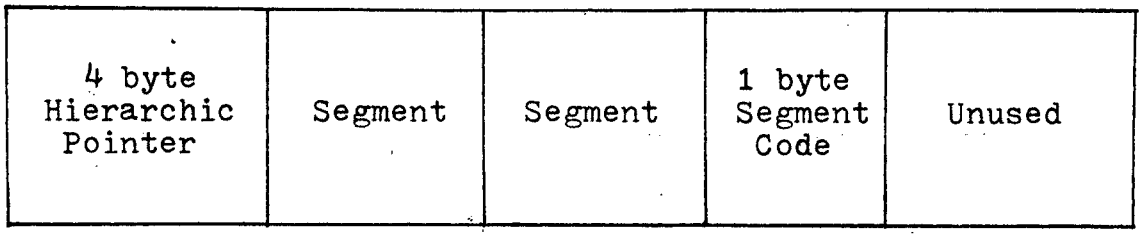

Fig. 4.2 Logical record of a VSAM implementation of a HISAM data base.

The four byte hierarchic pointer specifies the logical record which contains the next segment in hierarchic sequence. When new data base records are added after initialization they are inserted in their correct places in the KSDS data set and existing records are moved as necessary. Because of this there is no need to maintain root pointers.

In addition, VSAM logical records are stored in control intervals. Each control interval contains a fixed number of logical records (some of them unused) and ten bytes of VSAM

control information. Such an implementation minimizes the number of logical records which must be moved when a new record is inserted. The control intervals are maintained in B-tree fashion and the ten bytes of control information are used to maintain this B-tree.

With a HISAM data base, performance is affected by the choice of logical record length and by the choice of control interval length. These can affect both space requirements and access time. Figure 4.3 shows the effect of increasing the logical record length on the total space required for the data base and on the number of logical records required.

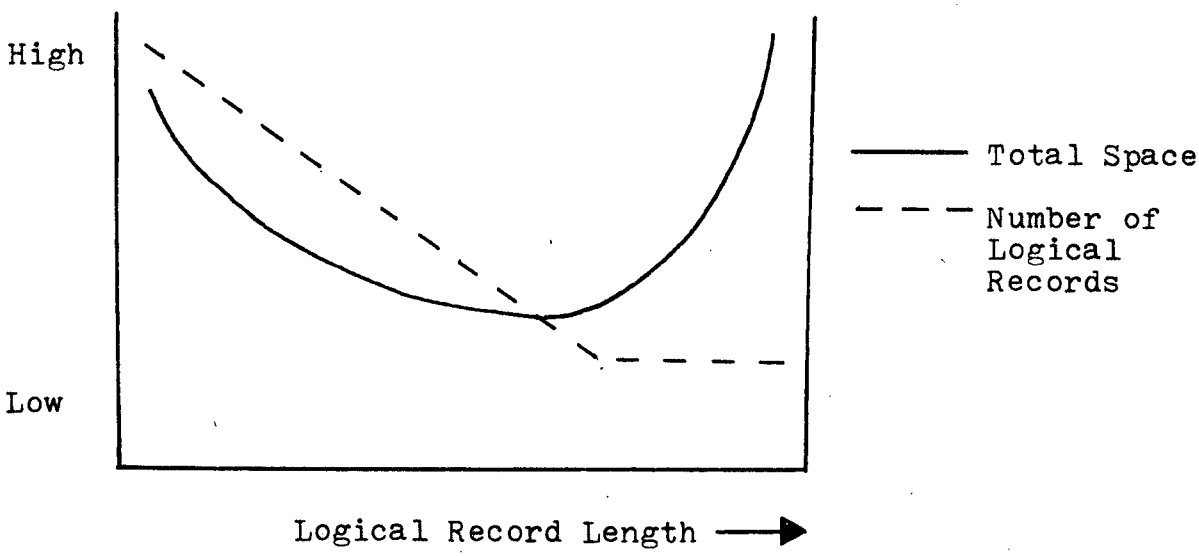

Fig. 4.3 Effect of logical record length on storage space.[13]

The logical record must be at least as long as the longest segment. As the logical record length increases, the

number of logical records decreases until the logical record length is as long as the longest data base record. At this point the number of logical records is equal to the number of data base records and the logical records curve goes flat. Correspondingly, the total storage space tends to rise if the logical record length is either too short or too long.

Changing the control interval size will have a similar effect; however, the storage space curve will be much steeper in this case as the overhead is greater. See Figure 4.4.

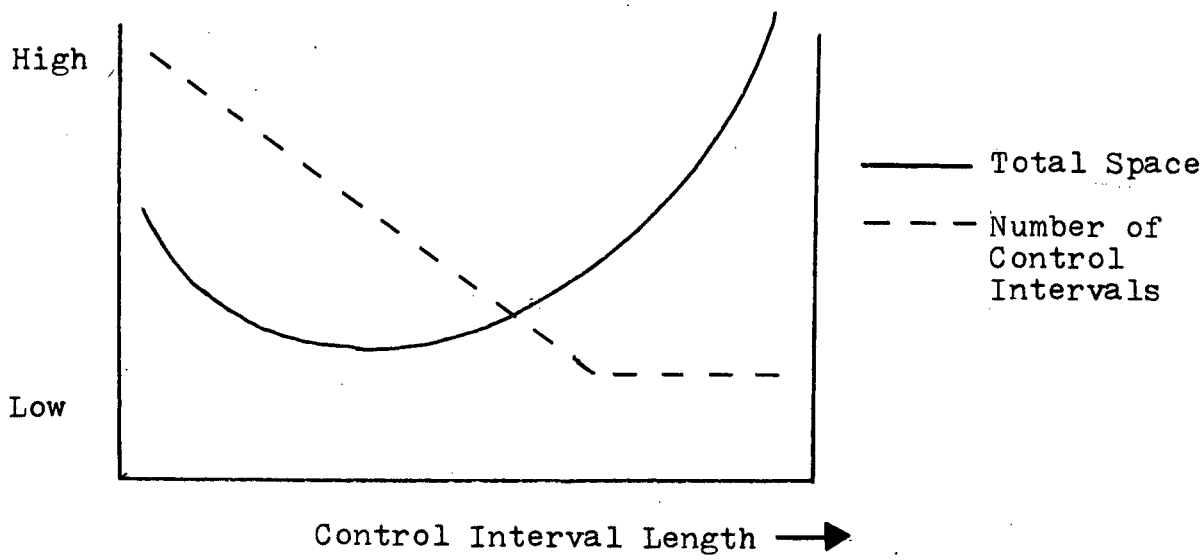

Fig. 4.4 Effect of control interval size on storage space.

In HISAM deletions are taken care of by setting a flag. This means that deleted segment space is not available for reuse (except in the case of root segments in the KSDS data set under VSAM). Processing can be done either sequentially,

by following the hierarchic sequence pointers for each root segment; or directly, by scanning the index data base for the appropriate root and then following the hierarchic pointers. It should be noted, however, that if a data base is very volatile (i.e., has a lot of insertions and deletions) performance can be severly degraded by the need to jump from the indexed area to the overflow area (and vice versa) when sequentially processing the data base.

For these reasons HISAM should be used for data bases in which both sequential and direct access by root segment is required and which do not require extensive segment insertion or deletion.

HIDAM

A HIDAM data base consists of two parts - the index data base and the data data base. As with HISAM these are implemented with KSDS and ESDS (for VSAM) or with HISAM/OSAM. In addition the data portion of the data base is composed of a root addressable area and an overflow area. See Figure 2.8.

The index data base contains one logical record for each data base record. Each logical record consists of an index segment for the root of the data base record and a pointer to the appropriate logical record in the ESDS or OSAM data set. In addition, if HISAM/OSAM is used, there is also a pointer to maintain the index sequence when new roots are added after

initialization. The index segment must be unique.

The entire data base record is stored in control intervals in the ESDS or HISAM data set. Each control interval contains a free space anchor point, free space elements, an anchor point area, segments, unused (free) space and (in the case of a VSAM implementation) seven bytes of control information. See Figure 4.5.

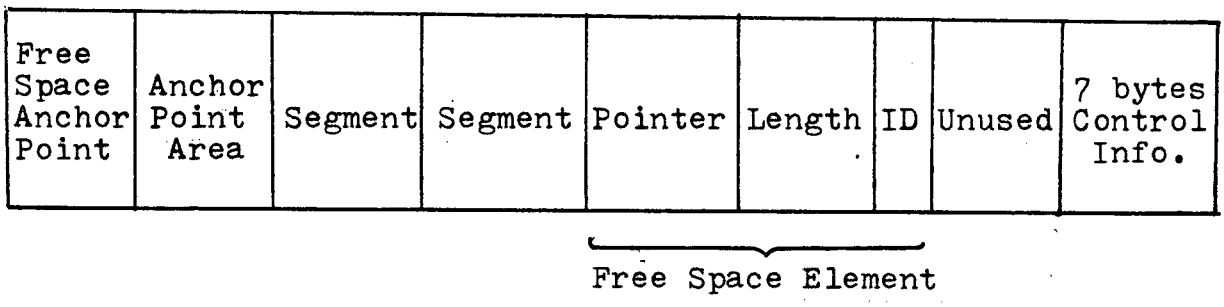

Fig. 4.5 Control interval of a HIDAM data base (7 bytes of control information for VSAM implementation only).

The free space anchor point is four bytes long and contains the offset in bytes to the first free space element in the control interval.

The free space element is used to identify each area of free space in the control interval which is eight bytes or more in length. This free space element is itself eight bytes long and consists of a free space chain pointer field, an available length field, and a task ID field to indicate which program freed the space. The anchor point area is four bytes

long and holds pointers to other control intervals to maintain the hierarchic sequence.

The first control interval of an OSAM data set or the second control interval of an ESDS (the first, in the latter case, is used for VSAM control information) is reserved for a bit map. This bit map is used to determine whether or not space is available in a particular control interval. Depending on the size of the control interval and the number of control intervals used it may be necessary to have several bit maps in order to cover the entire data base. In this case, subsequent bit maps are distributed throughout the data set and chained by an anchor point area at the first of each bit map control interval.

As with HISAM, an important performance consideration is the selection of an optimal control interval size. Also to be considered is the wise use of distributed free space (see Constraints and Extensions) which can help to minimize the amount of moving required when new segments are to be inserted.

Because of the chaining of free space and the ability to specify distributed free space, a HIDAM data base is able to reuse the space occupied by deleted segments and insert new segments in proximity to its position in the hierarchical sequence. Therefore, HIDAM is useful when both sequential and direct processing are required and when many insertions and deletions are probable.

HDAM

A data base using the HDAM access method will use the same storage structure as a HIDAM data base without the index data base. See Figure 2.7. HDAM uses a hashing function, instead of an index, to provide direct access to root segments. This hashing function may be either user supplied or one of many supplied by IMS.

When choosing a hashing function the user determines the size of the root addressable area. The user also specifies the maximum number of bytes of a data base record to be stored in the root addressable area; the remainder are stored in the overflow area.

The hashing function produces a control interval and an anchor point number. When multiple key values hash to the same control interval and anchor point, the synomyms are chained together in ascending key value sequence. The user may specify that from one to ten synonym chains be used per anchor point.

There are many ways in which the performance of a HDAM data base may be affected. A hashing function should be chosen which provides a good distribution of hash values to prevent the use of synonym chains whenever possible.

The size of the root addressable area should not be too small so as to increase the number of synonyms, nor should it be too large so as to have too much wasted space. The number of bytes of a data base record to be stored in the root addressable area should not be too small so as to make the

time required to follow pointers excessive nor should it be too long so as to waste space when the data base records are short.

Increasing the number of synonym chains will decrease the length of the individual chains as will increasing the number of anchor points or the size of the root addressable area. The shorter the synonym chains the less time will be spent following pointers.

The HDAM access method provides fast, direct access by eliminating the need to search an index. However, the key set should be relatively stable and must be capable of being randomized. It is possible to sequentially process the records as they are physically stored in the data set but, depending on the key and the hashing function used, this order is not necessarily the same as the root key order. This may make sequential access by root key order impossible. As for HIDAM, this method is good when many insertions and deletions are expected.

Space Considerations

One of the most important performance considerations is the amount of secondary storage the data base will require. Our ability to calculate accurately the exact amount of storage required is hampered by such things as variable sizes of segments, a variable number of dependents for any parent

and the general vicissitude of the data base. However, we can make approximations to the storage requirements under the different access methods and use these figures for comparative purposes when deciding which access method to use.

The following is an attempt to make these approximations. For each access method a formula will be derived relating the total storage requirements to the variables which apply to it. Table 4.1 contains a list of abbreviations used in these formulas and their meanings.

SR	Total storage requirement (bytes).
SR1	Storage required for the primary data set (bytes).
SR2	Storage required for secondary data sets (bytes).
WASS	Weighted average of segment size (bytes).
Si	Length of segment i (bytes).
Oi	Occurrences of segment i.
US	Unusable space (bytes/unit of storage).
BF	Blocking factor (bytes/block) or (bytes/interval).
LRL	Logical record length (bytes).
BLOCKS	Number of blocks required (blocks).
RECORDS	Number of logical records required (records).
CNTL	Number of control intervals required (intervals).
n	Total number of segments.
m	First segment in secondary data set.
k	Number of roots per control interval.
fbff	Free block frequency factor (0 to 100 excluding 1).
fspf	Free space percentage factor (0 to 99).
BMP	Storage requirement for bit maps (bytes).

Table 4.1 Abbreviations and descriptions.

The weighted average segment size (WASS) is the product of the length of each segment and the number of occurrences of that segment summed over all segments, divided by the total number of occurrences:

$$WASS = \frac{\sum_{i=1}^{n} SiOi}{\sum_{i=1}^{n} Oi}$$

The weighted average segment size of segments in the secondary data sets only (WASS') is given by the formula:

$$WASS' = \frac{\sum_{i=m}^{n} SiOi}{\sum_{i=m}^{n} Oi}$$

The total storage requirement (SR) is always the sum of the storage requirement for the primary data set (SR1) and for the secondary data sets (SR2):

$$SR = SR1 + SR2$$

For HSAM the total storage requirement is given by the amount of space needed to store the data plus a portion of each block which is left unused:

$$SR = \sum_{i=1}^{n} SiOi + US * BLOCKS$$

where:

$$US = \frac{WASS}{2},$$

$$BLOCKS = \frac{\sum_{i=1}^{n} SiOi}{BF - US}.$$

For HISAM (as for HIDAM and HDAM) the amount of storage required depends upon whether ISAM/OSAM or VSAM is used. For an ISAM/OSAM implementation there is one logical record in the index for each occurrence of a root segment:

$$SR1 = LRL * O_1$$

The secondary data sets are made up of the total space required to store the remaining segments plus the unused space at the end of each record:

$$SR2 = \sum_{i=m}^{n} SiOi + US * RECORDS .$$

where:

$$US = \frac{WASS'}{2} + 7 ,$$

$$RECORDS = \frac{\sum_{i=m}^{n} Si0i}{LRL - US} .$$

The VSAM implementation is slightly different. Since the control intervals are maintained like B-trees, on the average each will only be three quarters full. Each control interval also has ten bytes of control information:

$$SR1 = LRL * 0_1 * \frac{4}{3} + \frac{0_1}{k} * 10 .$$

Also, each logical record contains a five byte (vs. seven for ISAM/OSAM) pointer, which is considered unused space:

$$US = \frac{WASS'}{2} + 7 .$$

The other definitions remain the same as for ISAM/OSAM.

In the ISAM/OSAM implementation of HIDAM each index record contains the root plus a four byte pointer:

$$SR1 = (S_1 + 4) * 0_1 .$$

In the secondary data set there is a seven byte pointer in each control interval:

$$US = \frac{WASS}{2} + 7$$

and the total number of control intervals is:

$$CNTL = \frac{\sum_{i=1}^{n} SiOi}{BF - US} .$$

Each bit map is a control interval with eight bytes of control information. The number of bit maps required depends on the size of the control interval and the number of control intervals required to store the data:

$$BMP = \left(\frac{CNTL}{BF - 8}\right) * BF$$

the total storage requirements are then given by :

$$SR2 = \left(\sum_{i=1}^{n} SiOi + US * CNTL\right) * \frac{fbff}{fbff-1} * \frac{1}{1-\frac{fspf}{100}} + BMP$$

The significance of fbff and fspf will be introduced later.

With a VSAM implementation of HIDAM, each logical record in the index data base will only be three quarters full:

$$SR1 = (S_1 + 4) * O_1 * \frac{4}{3}$$

In the secondary data sets each control interval contains fourteen bytes of control information:

$$US = \frac{WASS}{2} + 14$$

And each bit map control interval has fifteen bytes of control information (there is also an extra control interval at the first of the data set):

$$BMP = \left(\frac{CNTL}{BF - 15} + 1 \right) * BF$$

The CNTL and SR2 definitions are the same.

The HDAM implementation will be the same as the HIDAM implementation without the index data base. So, for HDAM under ISAM/OSAM:

$$US = \frac{WASS}{2} + 7$$

$$CNTL = \frac{\sum_{i=1}^{n} SiOi}{BF - US}$$

$$BMP = \left(\frac{CNTL}{BF - 8}\right) * BF$$

$$SR = \left(\sum_{i=1}^{n} SiOi + US * BLOCKS\right) * \frac{fbff}{fbff-1} * \frac{1}{1-\frac{fspf}{100}} + BMP$$

and for a VSAM implementation:

$$US = \frac{WASS}{2} + 14$$

$$BMP = \left(\frac{CNTL}{BF - 15} + 1\right) * BF$$

Time Considerations

In order to make accurate time estimates of various queries under the different access methods it would be necessary to obtain a precise count of the queries and test them under the access methods or perform some type of

simulation[33]. Both of these alternatives are costly and time consuming, in fact, the former may be prohibitatively so and the latter lends itself to approximations and assumptions.

A "quick and dirty" method of evaluating the alternatives is presented. Table 4.2 relates each of the physical record interfaces associated with each access method to seven factors representative of modes of processing queries.

	Weight	HSAM SAM		HISAM ISAM/ OSAM		HISAM VSAM		HIDAM HISAM/ OSAM		HIDAM VSAM		HDAM OSAM		HDAM VSAM	
Update		1		3		4		5		5		5		5	
Insertion		1		3		2		4		4		5		5	
Deletion		1		5		5		3		3		4		4	
Retrieval		2		2		3		4		4		5		5	
Sequential Processing		3		3		3		5		5		2		2	
Direct Access Processing		2		3		3		4		4		5		5	
Reorganization		5		4		4		1		1		2		2	
Totals															

Table 4.2 Access method ratings for query response time.

Each access method is given a rating for each query type. A rating of 1 is poor while a rating of 5 indicates that this

is the best access method to use for this particular type of query. To evaluate the different access methods, a weight should be applied to each query according to how often such a query is expected and the importance the user places on its quick response. These weights are recorded in the first column and then applied to each of the ratings. The weight times the rating is recorded in the second column under the appropriate access method. These columns are then summed to provide a comparison of the different access methods. The arguments used in assigning these ratings were outlined earlier in this chapter when the access methods were considered in detail.

It may be beneficial to include storage costs in this same way. If storage costs are to be included, ratings can be assigned using the storage requirements for each access method and a new row can be added to the table. The user then assigns a weight to the storage requirements row and the totals calculated will take storage requirements into consideration.

Again, it should be noted that this method of comparison does not provide an absolute measure of the time required to satisfy a query. Rather, it provides a relative rating so that the various access methods may be compared in this respect.

Constraints and Extensions

When discussing the various methods so far, attention has been paid to only the basic attributes of each method; however, for each there are a number of constraints and extensions that may apply.

The constraints are summarized in Table 4.3. These will be the first eliminating factors when selecting an access method. Given that an access method is acceptable under these conditions, only then will further consideration be necessary.

Extensions exist in many forms, but four important ones are outlined here. Again, one should consult Table 4.3 to determine which access methods these apply to.

Distributed Free Space

In a data base in which there is much segment insert activity, performance can be severly degraded if inserted segments cannot be placed physically adjacent to their related segments. In order to alleviate this problem, a set amount of free (unused) space may be distributed throughout the data set at initialization.

Two factors are used to determine how much free space is set aside. The free block frequency factor (fbff) specifies that every nth control interval in the data set will be left as free space. The range of the fbff is from 0 to 100 excluding 1.

	HSAM SAM	HISAM ISAM/ OSAM	HISAM VSAM	HIDAM HISAM/ OSAM	HIDAM VSAM	HDAM OSAM	HDAM VSAM
Logical Relations Supported?	no	yes	yes	yes	yes	yes	yes
Secondary Indexes Supported?	no	yes	yes	yes	yes	yes	yes
Deleted Segment Space Reusable?	no	no	no	yes	yes	yes	yes
Distributed Free Space Factor?	no	no	no	yes	yes	yes	yes
Multiple Data Set Groups?	no	yes	no	yes	yes	yes	yes
Unique Key For Root Required?	no	yes	yes	yes	yes	no	no
Root Only Access Method Available?	yes	no	yes	no	no	no	no
Hierarchy Determines Physical Contiguity?	yes	yes	yes	no	no	no	no
Child-Twin Pointers Supported?	no	no	no	yes	yes	yes	yes
Insert, Delete and Replace Possible?	no	yes	yes	yes	yes	yes	yes
Direct Access to Segments Possible?	no	yes	yes	yes	yes	yes	yes

Table 4.3 Access method constraints.

The free space percentage factor (fspf) specifies the minimum percentage of each control interval to be left as free space. It can take on any value from 0 to 99. These factors have been incorporated into the storage requirement formulas for HIDAM and HDAM data bases.

It should be noted that although you decrease the access time when you increase the amount of free space, you do this at the expense of increasing the total amount of storage space required. It is important to choose values of fbff and fspf with this trade-off in mind.

Multiple Data Set Groups

Data bases often exist in which the lengths of the segments are quite different. In this case, logical record lengths which are optimal for the short segments may not necessarily be appropriate for the longer segments, or vice versa.

In such a case it may be appropriate to have one logical record length for some of the segments and another length for other segments. IMS/VS allows multiple data set groups. This means that the secondary dae broken up into as many as ten data set groups, each having its own logical record length.

A case of dramatic performance improvement by using multiple data set groups has been reported by Bruce Sicherman in the IMS Newsletter[34]. The most dramatic result was a 95% reduction in I/Os for a particular transaction. In other data bases, where multiple data set groups were effective, an

average elapsed time improvement of 33% was measured.

Although multiple data set groups can significantly decrease the amount of storage space and/or the number of I/Os required there is an increase in the overhead for maintaining the different data set groups. As well, the physical proximity of related segments might be affected. Again, a trade-off between space and time exists.

Root-Only Access Method

Two other access methods exist which have not yet been discussed. They are the Simple Hierarchic Sequential Access Method (SHSAM) and the Simple Hierarchic Indexed Sequential Access Method (SHISAM). In both cases the data base record may consist of only a single segment - the root. The prefix, which normally identifies the type of segment, is not needed.

Other than the above two distinctions, a SHSAM data base is stored similar to a HSAM data base and a SHISAM data base is stored similar to a HISAM data base. The saving comes in the form of both time and space. Less space is required because the prefixes are not needed and less time is needed to access a segment as there is only one type. However, SHSAM and SHISAM only apply to a very unique kind of data base and cannot normally be used.

Child-Twin Pointers

When pointers are used to access a data base they connect the segments in hierarchical sequence. To access a particular

child of a parent it is necessary to go through all the children which come before the desired child in hierarchical sequence. To provide more direct access to child segments, child-twin pointers may be specified.

When child-twin pcinters are used, a pointer to each child is specified in the pointer fields of the parent segment thus enabling you to go directly to the desired child. This is particularily important in data bases in which the segment types are processed randomly.

It should be noted that child-twin pointers can only be specified in the place of hierarchic pointers - the two kinds cannot exist together. The type of processing and the performance expected from each kind must be considered before one of the two is selected.

Chapter 5 - Other Performance Considerations

Introduction

In a system as large as IMS there exists a great number of factors which, in one way or another, determine the performance of the system. In the previous two chapters some of the more prominent issues (i.e., those which are thought to have the greatest effect on performance) have been discussed. It is not possible to present here a complete enumeration of all relevant factors with their various merits, vices and interactions. Such a study is beyond the scope of this paper and may well be impracticable.

For the sake of completeness three more topics will be covered in some detail - buffering facilities, application programming and main storage requirements. Each of these will be covered in its own section later in this chapter.

In addition to these, there are some other matters which bear mentioning. These latter topics - the fast path feature, user exit routines and data base reorganization - will be introduced here with what is hoped to be sufficient references to satisfy the interested reader.

Fast Path

The fast path feature is an alternate transaction processing mode. It is suitable for applications which require good response characteristics and which may have large transaction volumes but which only need simple data base

structures. Fast path runs in conjunction with the IMS/VS data communication facility and does not replace it. Data bases using the fast path feature can be accessed by either fast path or IMS/VS transactions.

For more information on the fast path feature see the Fast Path Feature General Information Manual[14]. In addition, for design considerations see Chapter 6 of the System/Application Design Guide[20] and Chapters 1 - 5 of the System Programming Reference Manual[21].

Exit Routines

IMS provides for the use of exit routines. These routines may be either written by the user or drawn from a library of routines supplied by IMS. Exit routines allow the use of internally generated data as well as permitting users to incorporate processing extensions of their own. Of particular interest (with regard to performance) is the Segment Edit/Compression Exit which permits the editing of a segment during its movement between the data base buffer pool and the input/output of the application program. Thus, one can compress data for storage and expand it again for processing.

User exit routines are covered in Chapters 3 and 4 of the System Programming Reference Manual[21]. Some design considerations are given in the System/Application Design Guide[20] pp. 4.125 - 4.148.

Reorganization

Data base reorganization is a topic which must be considered in an application which is in any way volatile. Reorganization is the act of restructuring a data base to reflect changes in processing or to improve performance in a data base in which utilization of space and/or contiguity of segments has been degraded due to a number of inserts and deletes. The actual and CPU time required for such an operation is usually substantial and must be weighed against the improved performance expected from the reorganization.

Reorganization of IMS data bases is discussed in Chapter 5 of the Utilities Reference Manual[22] and more generally in the System/Application Design Guide[20] pp. 4.150 - 4.152. For a discussion of the principles of reorganization see "Restructuring for Large Data Bases: Three Levels of Abstraction"[28] and "Database Reorganization - Principles and Practice"[35].

Buffering Facilities

In IMS/VS pools of buffers are maintained to decrease the required number of accesses to secondary storage by allowing blocks of data to remain in virtual storage as long as possible. Three types of buffer pools exist - the ISAM/OSAM buffer pool, the VSAM shared resource pool and the DL/I buffer handler pool.

The DL/I buffer handler pool is used in conjunction with an ISAM/OSAM buffer pool or a VSAM shared resource pool depending on the access method of the data base in use. It is used for recording buffering services activity. This includes statistics on buffering services as well as recording calls for buffering services, open and close of data bases and program isolation enqueues and dequeues.

Data bases that use the VSAM access method share the use of buffers in the VSAM shared resource pool. Data bases that use the ISAM or OSAM access method share the use of buffers in the ISAM/OSAM buffer pool. Buffers are combined in subpools; all buffers within a subpool being of the same length. Buffers are chained together in a use chain. Empty buffers are placed at the bottom of the chain and are always ready for use. As buffers are accessed they are placed at the top of the chain.

When a data set is opened it is assigned a buffer subpool which contains buffers at least as large as the data set block or control interval size. When a retrieval request occurs, the buffer pool is searched, using the use chain, to see if the required segment is in virtual storage. If it is found, the appropriate buffer is placed at the top of the use chain. If it is not found, the buffer at the bottom of the chain (the least recently used) is selected. If it contains old data and that data has been changed, the buffer is written out. The requested data is then read into the buffer and the buffer is moved to the top of the use chain.

The number and size of each kind of buffer pool is determined by the user. A time/space tradeoff exists here which must be reconciled to provide acceptable performance for an installation. Providing a variety of buffer subpools and increasing the total number of buffers will improve the response time as less I/O will be required to satisfy each DL/I request. However, this also means that more virtual storage is taken up by the buffers - some of which may not be used at all times.

If it is necessary to use buffers which are larger than the block or control interval size some space will be wasted at the end of each buffer. The best situation is to have the buffer size equal to the block or control interval size but this may limit the number of data sets which may use a particular subpool and thus have the whole subpool go to waste when no appropriate data set is in use.

A data base with high activity may tend to overload a particular subpool. This may be avoided by defining the data set block or control interval size in such a way that the data base is assigned a unique subpool. This will relieve the strain on other subpools and reduce contention among data sets.

When determining which data sets will share subpools it is important to consider how the application programs will access the data bases. For example: Are segments generally requested once-only? Is retrieval of records in hierarchical or some other order? Are a certain percentage of segments

always requested? Contention can be reduced by grouping together data bases that have similar processing requirements or which never require the same subpool at the same time.

Application Programming

One of the greatest criticisms of IMS is that it imposes a hierarchical structure on the data and that it requires a programmer to be familiar with at least part of this structure if she/he is to write programs which successfully perform transactions. To go one step further, a more comprehensive knowledge of both the data structure and DL/I processing is necessary if these transactions are to be carried out in an efficient manner.

By being aware of the order in which the data base is stored the application programmer can greatly reduce the number of I/Os by processing the data base in that order whenever possible. The existence of secondary indexes or the possibility of introducing such indexes should be made known to all application programmers. Facilities such as these, which are designed for a specific application, may have a use in another application as well.

When processing data bases it is best to retrieve a root segment and process all segments of that record before moving on to another root or another data base. Since a root and its child segments can typically be retrieved with one access,

time is saved by doing as much processing as possible on a given record when it is in virtual storage.

Search time can be significantly decrease by making all DL/I calls as fully qualified as possible. This entails specifying a value for each key of each segment on the hierarchical path to the desired segment. This reduces the number of paths to search and thus eliminates some of the overhead.

When a segment is to be retrieved it is necessary to issue a DL/I 'get' call (see Table 2.2 for a summary of DL/I subroutine calls and descriptions). If this segment is to be changed in any way a 'get hold' call is required. Because there is very little performance difference between a 'get' and a 'get hold' call, the 'get hold' should be used whenever there is a reasonable chance (about 5% or more) that the segment will be changed. This eliminates the need to do both a 'get' and a 'get hold'.

When inserting and deleting fields or segments it is important to bear in mind the policy implemented by the access method being used. The insertion or deletion of longer or shorter fields or segments may have adverse effects on performance. These policies have been discussed in Chapter 4 under the appropriate access methods. Whenever possible, it is always better to use the 'replace' call over a 'delete' followed by an 'insert' as this eliminates the need to make changes to pointers and reduces space management overhead.

With each DL/I call one or more optional command codes

may be specified. Of particular interest is the D command code (for a complete treatment of comand codes see chapter 3 of the Application Programming Reference Manual[15]). The D command code allows the issuance of path calls. A path is a hierarchical sequence of segments, one per level, leading from a segment at a higher level to one at a lower level. A path call enables a hierarchical path of segments to be inserted or retrieved with one call.

"The correct usage of path calls can have a significant performance advantage. You should use it whenever possible, even if the chance of the existence of the need for the dependent segment(s) is relatively small. For instance, if you would need, in 10% or more of the occurrences, the first dependent segment after you inspect the parent, then it is generally advantageous to use a path call to retrieve them both initially. The first dependent segment (in this case) is retrieved at almost no additional cost."[14]

Main Storage Requirements

When a system as large as IMS is installed there is bound to be an increase in the amount of main and secondary storage required. The secondary storage requirements for the data bases themselves was discussed in Chapter 4. In addition to this the amount of main storage required is determined by the users' application and environment. For example; the number of data bases, the number of users of each data base and the number of remote terminals will all have an effect on the

total main storage requirements.

In some respects, the amount of main storage required is determined by what is considered by the user to be "acceptable" performance levels and which trade-offs are made to insure that these performance levels are met. For example, decisions to use secondary indexes, logical relationships and fast path processing will each affect the main storage requirements.

In other respects, a certain amount of main storage is required to support the basic IMS/VS system - independent of the application, environment or selected optional features.

The developers of a new IMS installation should be aware of the demands that the adoption of the system will have on main storage. Similarily, when an existing IMS system is being tuned the effect of proposed improvements on the main storage requirements should be known before a final decision is made.

Fortunately, these storage requirements can be easily and accurately estimated. Chapter 5 of the Systems Programming Reference Manual[21] provides a detailed description of these calculations including worksheets and examples; for this reason they will not be duplicated here. It is sufficient for our purposes to note that such a procedure for making accurate estimates exists as these estimates will be useful in a thorough performance study.

Chapter 6 - Applications

Introduction

The role of performance evaluation at a particular installation may be vastly different from that at another installation. The needs of the user community, the data processing history of the enterprise, the type of application and the actual system configuration all combine to determine the number and kind of performance studies to be carried out in the organization. These studies may range from ad hoc measurements for locating bottlenecks when the system is responding poorly to sophisticated monitors which provide feedback for optimizing file organizations and restructuring data bases.

In this paper we have considered ways in which the performance of an IMS installation may be improved. The question of the practicality of such methods remains unanswered. There may be a large gap between what in theory seems to be justified and what in reality is actually done. This is not to say that the theory is wrong nor that the method is correct. What is important, in this case, is that solutions to performance problems be effective and implementable.

In order to obtain a 'real world' perspective, systems personnel from three large IMS installations were interviewed. The remainder of this chapter is based upon those interviews. This is not meant to be taken as a statistical sample nor

would it be appropriate to draw firm conclusions at this point, however, they do serve to point out the kinds of things which are being done with regard to performance and may indicate the appropriateness of the performance issues presented in this paper.

Case 1: B. C. Hydro

The IMS application at B. C. Hydro consists of one large data base with over 60 segments. This is in contrast to the arguments presented in Chapter 4 which suggest that data bases are more efficient if they are smaller. In addition, Chapter 3 recommends that data bases be designed with a specific relationship among the segments, which, in this case, would imply several smaller data bases. IMS was first adopted there in 1971 and at that time little was known about IMS and less about the effects of design on performance.

The first implementation used HISAM as the access method primarily because it was well known and had been in use for several years at many other installations, little was known about the other 'new' access methods. However, because the data base was very volatile and a large number of segments ended up in the secondary data set, HIDAM was adopted to reduce the switching from primary to secondary data sets when sequentially processing the data base. It was later realized that it was never really necessary to process the data base in

root key sequence and hence the index was being maintained uselessly. Because of that HDAM is now being used.

As can be seen here, the importance of doing a thorough analysis of the enterprise's data processing needs is an integral part of the data base design. This point was made in Chapter 3 and this example illustrates some of the possible repercussions.

There is an unofficial policy at B. C. Hydro to keep the data base simple. Logical relationships and secondary indexes are not being used because it is thought that things such as these will overtax IMS and lead to unnecessary delays due to increased processing demands. In fact, much of what could be done by IMS is done by special macros or within the application programs. The reasoning behind this is that IMS is a very generalized system, developed to handle as many cases as possible. Therefore, if you write the code yourself, with only one purpose in mind, you eliminate the overhead necessary to look after a variety of cases.

Although this idea of replacing or supplementing some IMS tasks has not been discussed previously, it seems to be a viable solution to some specialized problems. Difficulities may arise, though, if the macros make use of IMS internals - especially when a new release of IMS is issued.

Performance studies at B. C. Hydro take the form of ongoing exception reporting. Statistics are routinely collected but a performance study is really only carried out when response time is degraded or disk space is at a premium.

Although the data base is very volatile, the total number of segments does not vary by much. The data processing requirements are relatively the same from year to year. For these reasons, and because HDAM is now being used, reorganization of the data base is infrequent and modifications to the data base structure amount to changes to only a few segments per year. This seems to be in agreement with what would be recommended by using Table 4.2.

Case 2: I. C. B. C.

The Insurance Corporation of British Columbia has been using IMS to look after its data processing needs since 1974. There are over 40 separate data bases in all, dealing with various facets of employee and customer relations. Each data base has relatively few segment types and reflects a specific relationship among entities as was recommended in Chapter 3.

Most data bases are implemented under HDAM but HIDAM, HSAM and SHISAM are also used. When an index is required for a data base, VSAM is always used. When HDAM is used, all data base records are root addressable so that it is never necessary to use overflow chains. This is possible because the root keys are evenly distributed. This distribution is monitored and the randomizing routine changed if necessary. The importance of using an appropriate randomizing routine and of reducing the useage of overflow chains was outlined in

Chapter 4 and seems to be supported here.

I. C. B. C. handles insurance policies for the whole province of British Columbia. This makes it necessary to support over 200 user terminals and supply them with good response times. Over the past three years they have been able to decrease the average response time from 12 to 2 seconds, despite the fact that the load has increased 10 times to an average of 60,000 on line transactions per day at present.

Both logical relationships and secondary indexes are used and significant performance improvements have been gained by their implementation. The use of presorts to arrange the transactions into the same order as the stored records was also found to be beneficial in many circumstances. Macros are utilized to provide services not available through IMS and in some cases to replace inefficient IMS routines.

At I. C. B. C., considerable attention is paid to performance monitoring and improvement. In particular, two packages - Data Base Monitor and Data Communications Monitor - are run every day to provide a snapshot report of the usage of the data bases and communications network. Another monitor - Control IMS - is continually recording buffer pool usage, queue sizes and terminal activity. The advantages of using such monitors was discussed in Chapter 2.

To improve performance, consideration is also given to such hardware features as disk and channel speeds and the size of main memory. Because on line service is provided to remote stations in all of British Columbia, the way in which these

terminals are connected to the central computer has a significant effect on performance and therefore must also be taken into consideration. These topics are beyond the scope of this paper but in many cases prove to be the bottlenecks to improved performance.

Case 3: B. C. Tel

IMS has been in use at B. C. Tel since 1976. There are actually three different IMS systems in operation: one for production, one for development and one for testing. The production and testing systems run on the same machine with other (non-IMS) applications. A total of 25 different data bases have been developed and approximately 700 terminals all over British Columbia are in use for on line processing.

HISAM, HIDAM, and HDAM are all being used with VSAM as the processing routine. The significant performance advantage of using VSAM over ISAM/OSAM has not been articulated in the previous chapters but seems to be evident from all three of these organizations. Both logical relations and secondary indexes have been utilized to create increased flexibility and ease of access despite the fact that they degrade performance. Such tradeoffs, as discussed in Chapter 3, were expected.

The B. C. Tel Customer data base occupies four 3350 disks and absorbs hundreds of changes a day. It experiences a 10 to 15% growth rate each year, but because VSAM is being used this

does not create problems and additicnal space is acquired as necessary. The Toll data base which records customer charges, must be reorganized every two weeks requiring about 6 hours of elapsed time. However, this data base is kept in multiple data sets which are reorganized individually, cutting down on the inconvience and time required to reorganize. The advantages of using multiple data set groups in cases such as this were presented in Chapter 4.

Performance is monitored on a daily basis with the Data Communications Monitor and with the System Log. These provide information on buffer usage, I/O waits, page faults, transaction rates, data base calls, etc. As was discussed in Chapter 5, the selection of correct buffer sizes can have a significant effect on performance. Although these do not usually precipitate data base or system changes they are useful when a new application is being developed. When problems do arise, a trial and error approach is used to reconcile the situation although with some problems the causes (and thus the solutions) are more easily perceived.

At B. C. Tel, a strong emphasis is placed on a thorough analysis of such things as transaction types and volumes, expected response times, call sequences, etc., before a data base is designed for a new application. Because of this, new applications are brought on line with few problems. Emphasis is also placed on incorporating plans for future applications into designs whenever possible.

Conclusions

In this paper an effort was made to pinpoint those aspects of IMS which affect system performance. In addition to this, three IMS installations were examined to test the ability of such aspects to regulate performance.

It was found that the topics covered here are those which are important from a performance stand point. Of these, the choice of access method will probably have the greatest effect. Where possible, HDAM should be used. HIDAM should be adopted if an index is necessary. VSAM is the most efficient physical record interface and its additional storage requirements are easily justified by its ability to reuse space.

It is important to distribute free space throughout the data sets, but determining the correct amount is dependent upon the application and in particular, the frequency of update. Multiple data set groups are useful when the size of the segments within a data base vary considerably or when the data base has very few roots, each with a large quantity of children. Otherwise, they will increase the amount of storage required and actually degrade the response time.

Logical relationships and secondary indexes may be regarded as a retrogression. They do, however, simplify application programming and may be considered a performance improvement in the long run. Incorrect or inappropriate usage of these can have severe negative effects.

While the design of an IMS data base can affect

performance, it is more likely to affect the facility of developing application programs than it is to affect response times. The processing order should be kept in mind and segments should be as close to the same length as possible.

Changes to such things as the number and size of buffer pools, control interval lengths and blocking factors can produce noticible improvements. The need for a change in these areas is usually detected by examining the system log or using some other monitoring tool. The degree of change is usually determined by experimentation.

IMS is a very large and generalized data base management system. Because requirements and applications change from installation to installation there will often be more efficient ways of doing things than would normally be done by IMS. These may be used to augment IMS but should never duplicate services that can be done better by IMS nor should they attempt to manage the data that IMS has control over.

In many installations there may be other factors which have an overriding effect on performance. For example, in a system with remote terminals, where much of the response time is taken up with data transfer, it will be more beneficial to look at ways of improving the line speed than attempting to reduce the processing time.

Things such as the interface between IMS and the operating system, network characteristics, the use of reenterant and reusable code and hardware characteristics have not been treated here. This is not to say that they are not

important, on the contrary, these are often more important, but fall into a different class of performance study.

In conclusion, it is my opinion that a more rigorous approach to performance evaluation at the organization level is needed. Performance improvements seem to be done haphazardly and more by intuition than by reasoning. Tools such as queing theory and simulation have applications here and could be used at both the design and tuning stages of system development. Techniques of system analysis are necessary to insure that the users needs are being satisfied and that the system meets these needs in an efficient manner.

Footnotes

1. CODASYL Systems Committee, "Selection and Acquisition of Data Base Management Systems," (March 1976), p. 4.

2. Datapro 70 The EDP Buyers Bible, Vol. 3 (Deban, N.J.: McGraw-Hill, April 1979), p. 70E-010-61A.

3. Ibid. p. 70E-010-61K.

4. Domenico Ferrari, Computer Systems Performance Evaluation (Englewood Cliffs, N. J.: Prentice-Hall Inc., 1978), p. 2.

5. IMS/VS Version 1 General Information Manual, Release 1.5, 9th. ed. (San Jose: IBM Corp., April 1979) p. 2.37.

6. D.C. Tsichritzis and F.H. Lochovsky, "Designing the Data Base," Datamation Vol. 24, No. 8 (Aug. 1978) p. 147.

7. E.M. Gearhart, "IMS Data Base Design," Proceedings of SHARE 52 (March 1979) p. 365.

8. D.C. Tsichritzis and F.H. Lochovsky, "Designing the Data Base," Datamation Vol. 24, No. 8 (Aug. 1978) p. 147.

9. John K. Lyon, The Data Base Administrator (New York: John Wiley and Sons, 1970) p. 21.

10. IMS/VS Version 1 Primer, Release 1.5, 1st. ed. (San Jose: IBM Corp., Sept. 1978) p. 2.78.

11. S.B. Yao and A.G. Merten, "Selection of File Organization Using an Analytic Model," ACM Transactions on Database Systems Vol. 1, No. 2 (June 1976) p. 256.

12. Alfonso F. Cardenas, "Evaluation and Selection of File Organization - A Model and System," Communications of the ACM Vol. 16, No. 9 (Sept. 1973) p. 547.

13. IMS/VS Version 1 Systems/Application Design Guide, Release 1.5, 7th. ed. (San Jose: IBM Corp., Sept. 1978) p. 4.25.

14. IMS/VS Version 1 Primer, Release 1.5, 1st. ed. (San Jose: IBM Corp., Sept. 1978) p. 4.22.

Bibliography

1. "A Buyers Guide to Data Base Management Systems." Datapro 70 The EDP Buyers Bible, Vol. 3. Deban, N. J.: McGraw-Hill. April 1979.

2. Aho, Alfred L., John E. Hopcroft and Jeffrey D. Ullman. The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison-Wesley Publishing Co., 1974.

3. Benwell, Nicholas. Benchmarking: Computer Evaluation and Measurement. Washington, D.C.: Hemisphere Publishing Corp., 1974.

4. Cardenas, A.F. "Evaluation and Selection of File Organization - A Model and System." Communications of the ACM, Vol. 16, No. 9 (Sept. 1973) pp. 540-548.

5. "Data Base Design Methodology - Part I." Data Base Management. Philadelphia: Auerbach Publishers, Inc., 1976.

6. Date, C.J. An Introduction to Database Systems, 2nd. ed. Reading, Mass.: Addison-Wesley Publishing Co., 1977.

7. DB/DC Data Dictionary General Information Manual, Release 3.0, 3rd. ed. GH20-9104-2. San Jose: IBM Corp., Oct. 1978.

8. Dearnley, P. "Monitoring Database System Performance." The Computer Journal, Vol. 21, No. 1 (Feb. 1978) pp. 15-19.

9. Drummond, M.E. Jr. "A Perspective on System Performance Evaluation." IBM Systems Journal, Vol. 8, No. 4 (1969) pp. 252-263.

10. "Establishing a Framework for Data Base Planning." Data Base Management. Philadelphia: Auerbach Publishers Inc., 1976.

11. Ferrari, Domenico. Computer Systems Performance Evaluation. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1978.

12. Gearhart, E.M. "IMS Data Base Design." Proceedings of SHARE 52. SHARE Inc. (1979) pp. 350-380.

13. IMS Application Development Facility Program Description/Operators Manual, Version 1, Release 2, 4th. ed. SG20-1931-3. San Jose: IBM Corp., Nov. 1978.

14. IMS Fast Path Feature General Information Manual, 3rd. ed. GH20-9069-2. San Jose: IBM Corp., April 1978.

15. IMS/VS Version 1 Application Programming Reference Manual, Release 1.5, 7th. ed. SH20-9026-6. San Jose: IBM Corp., Sept. n.m.

16. IMS/VS Version 1 General Information Manual, Release 1.5, 9th. ed. GH20-1260-8. San Jose: IBM Corp., April 1979.

17. IMS/VS Version 1 Master Index and Glossary, Release 1.5, 4th. ed. SH20-9085-3. San Jose: IBM Corp., Dec. 1978.

18. IMS/VS Version 1 Operators Reference Manual, Release 1.5, 7th. ed. SH20-9028-6. San Jose: IBM Corp., Sept. 1978.

19. IMS/VS Version 1 Primer, Release 1.5, 1st. ed. SH20-9145-0. San Jose: IBM Corp., Sept. 1978.

20. IMS/VS Version 1 Systems/Application Design Guide, Release 1.5, 7th. ed. SH20-9035-6. San Jose: IBM Corp., Sept. 1978.

21 IMS/VS Version 1 Systems Programming Reference Manual, Release 1.5, 8th. ed. SH20-9027-7. San Jose: IBM Corp., Oct. 1978.

22. IMS/VS Version 1 Utilities Reference Manual, Release 1.5, 7th. ed. SH20-9029-6. San Jose: IBM Corp., Sept. 1978.

23. King, John Leslie and Edward L. Schrems. "Cost-Benefit Analysis in Information System Development and Operation." Computing Surveys, Vol. 10, No. 1 (March 1978) pp. 19-34.

24. Lochovsky, F.H. and D.C. Tsichritzis, Data Base Management Systems. New York: Academic Press, 1977.

25. Lyon, John K. The Database Administrator. New York: John Wiley and Sons, 1970.

26. McGee, W.C. "On User Criteria for Data Model Evaluation." ACM Transactions on Database Systems, Vol. 1, No. 4 (Dec. 1976) pp. 370-387.

27. McGee, W.C. "The IMS/VS System." IBM Systems Journal, Vol. 16, No. 2 (1977).

28. Navathe, S.B. and J.P. Fry. "Restructuring for Large Data Bases: Three Levels of Abstraction." ACM Transactions on Database Systems, Vol. 1, No. 2 (June 1976) pp. 138-158.

29. Nunamaker, J.E. and Benn R. Konsynski, Jr. "Computer Aided Analysis and Design of Information Systems." Communications of the ACM, Vol. 19, No. 12 (Dec. 1976) pp. 674-687.

30. Raver, N. and G.U. Hubbard. "Automated Logical Data

Base Design: Concepts and Applications." IBM Systems Journal, Vol. 16, No. 3 (1977) pp. 287-312.

31. Ross, Ronald G. "Evaluating Data Base Management Systems." Journal of Systems Management, Vol. 27, No. 1 (Jan. 1976) pp. 30-35.

32. "Selection and Acquisition of Data Base Management Systems." CODASYL Systems Committee. March 1976.

33. Senko, M.E., V.Y. Lum and P.J. Owens. "A File Organization Evaluation Model (FOREM)." IFIP Congress (1968) pp. 514-519.

34. Sicherman, Bruce. "Dramatic Performance Improvement With a Simple Technique." IMS Newsletter, Vol. 3, No. 3 (Nov. 1979).

35. Sockret, G.H. and R.P. Goldberg. "Database Reorganization - Principles and Practice." Computing Surveys, Vol. 11, No. 4 (Dec. 1979) pp. 371-395.

36. Thompson, Steve. "An Experience in IMS/VS Tuning." IMS Newsletter, Vol. 3, No. 3 (Nov. 1979).

37. Tod, Mary Kathleen. "Performance Considerations in Relational and Hierarchical Data Base Management Systems." M.Sc. Thesis. U.B.C. Feb. 29, 1980.

38. Tsichritzis, D.C. and F.H. Lochovsky. "Designing the Data Base." Datamation, Vol. 24, No. 8 (Aug. 1978) pp. 147-151.

39. Welsh, Myles E. "Getting Ready for IMS/VS." Datamation, Vol. 24, No. 13 (Dec. 1978) pp. 109-118.

40. Wiederhold, Gio. Database Design. New York: McGraw-Hill Book Co., 1977.

41. Yao, S.B., K.S. Das and T.J. Teorey. "A Dynamic Database Reorganization Algorithm." ACM Transactions on Database Systems, Vol. 1, No. 2 (June 1976) pp. 159-174.

42. Yao, S.B. and A.G. Merten. "Selection of File Organization Using an Analytic Model." Proceedings of the International Conference on Very Large Data Bases (Sept. 1975) pp. 255-267.