PERFORMANCE OF VIDEBAS IN AN OPERATIONAL ENVIRONMENT

H. M. BLANKEN

Twente University of Technology, Department of Computing Science, P.O. Box 217,
 7500 AE Enschede, The Netherlands

(Received 4 March 1983; in revised form 13 June 1983)

Abstract—VIDEBAS is a relational database management system in which a database consists of two parts, namely a “real-only” and an “update” part. The first part remains unmodified until the next reorganization and exploits redundancy to achieve fast access to data. A prototype of VIDEBAS has been built. In this paper a performance comparison between this relational system and a DBTG-system (UDS) is made. The used external memory and the number of page accesses to retrieve and update tuples is estimated. Although it is commonly assumed that in an operational environment relational systems are slower than network systems the opposite appears. On the other hand UDS needs less external memory.

1. INTRODUCTION

Information systems are often divided into strategic and operational systems. The strategic user is retrieval oriented and mostly interested in trends. So data which are not completely up-to-date may be acceptable and retrieval of data is more “set-oriented”. In the operational environment, on which we concentrate from now on, up-to-date data are necessary and the access pattern is more “tuple-like”. We confine ourselves to environments which have a not too high update frequency (see also Section 3.3) and which allow periodically closing of the database (for instance each night or weekend).

An operational system can be built upon a relational system as well as upon a network system. It is commonly assumed that, at least in the operational environment, relational systems are slower than network systems.

The architecture of the relational system VIDEBAS has been described in[1]. A VIDEBAS database consists of two parts, one containing data which are not modified until the next reorganization (the “read-only” part), the other comprises the updates (the “update” part). The read-only part uses data redundancy to achieve speed. So external memory is sacrificed to get faster access. This is justified by the observation that during the last two decades random access devices increased their storage capacity with a factor of about hundred, while in the area of speed only a factor of three to four is achieved. The update part is stored in a special, fast memory. This memory can be implemented by CCD and/or bubble memories. In the sequel no “special” hardware is assumed however; the update part is located in a program which is stored in (virtual) memory. A prototype of the VIDEBAS system is running on the DEC-system 10 and is described in[2]. Until now it has been tested only for a very small database.

UDS is a DBTG-system which is operational already for several years. At the university of Darmstadt, UDS has been studied extensively, especially in the area of performance, see[3, 4].

UDS and VIDEBAS are described, as far as needed for the comparison, in chapter two, while chapter three treats the used models. Chapter four estimates and compares the required external memory, while in chapter five the number of page accesses for a certain load is computed. Chapter six concludes the paper with some remarks.

2. DESCRIPTION OF THE SYSTEMS

2.1 UDS database system

In Fig. 1 the so-called independent UDS system has been characterised very briefly, see[3]. The application programs (AP) and the DBTG system (UDS) communicate to exchange DML statements and results. The UDS system accesses the database. The database is paged and the unit of transfer is a page. In UDS a bufferpool is present in which pages can be stored. When a page is needed, first the bufferpool will be searched and if this is not successful the database is accessed.

In UDS a SEARCH KEY (=an index) can be defined for a record type as well as for a set type. The

Fig. 1. UDS architecture.

latter implies that each set occurrence gets an index. Indexes in UDS are B*-trees. A set is implemented as POINTER-ARRAY, CHAIN or LIST. A POINTER-ARRAY consists of a sequence of POINTERS to the member records of the set. CHAINING implies that each member record is connected by means of an imbedded POINTER to the next and (sometimes) to the previous member of the set. The owner contains POINTERS to the first and (sometimes) last member of the set. A LIST consists of a sequence of records belonging to the set. Members contain always the Data Base Key of the owner. Each record in a DBTG database has a Database Key, which gives access to an entry in the Data Base Translation Table (DBTT). Each record type has a DBTT and for each record occurrence an entry exists. Besides the physical address of the record, an entry contains physical addresses of POINTER-ARRAYs, LISTs and SEARCH KEYS of sets owned by this record. A POINTER consists of two parts, namely the Data Base Key and physical address of the initial position of the record. Given a POINTER the physical address points to a record as long as the record stays on the initial page. When a record is moved from the page on which it is stored then only the DBTT entry will be updated. Access to a record given a POINTER will then be achieved via the Data Base Key, which gives the right DBTT entry (see Fig. 2).

The SET ORDER clause determines the insertion point in a set. Many possibilities are offered in UDS of which only two are considered, namely SORTED INDEXED and IMMATERIAL. SORTED means that the set members are logically ordered on a given attribute. INDEXED implies an index for each set to allow fast direct access for the concerned attribute. IMMATERIAL (the other SET ORDER) allows the UDS system to choose a convenient way of inserting the new member in the set.

Hashing is available in UDS (LOCATION MODE IS CALC) and pages with hashed records only contain records of that type.

There is a free page administration (FPA) which contains the administration of the free space for each

page in the database. This FPA is needed for STORE, ERASE and other DML statements.

2.2 VIDEBAS database system

The read-only part of a VIDEBAS database remains unchanged until the next reorganization. All updates to the database are collected in differential files, one file for each relation. To achieve a good performance it is important that the operations on the differential file are executed fast. The size of the update part may be a few megabytes. The prices of MOS memories and Winchester disks drop very fast, suggesting the storage of differential files in virtual memory, which is more or less a combination of those two memory technologies. In Fig. 3 the architecture of the VIDEBAS implementation has been shown.

2.2.1 The Architecture of the implementation.

Storage model. A relation may be stored many times, each time ordered on (a) different attribute(s). Such a file is called a SORTFL. Each SORTFL has a directory (DIRFL) which contains information to allow direct access to the SORTFL given (a) value(s) of the attribute(s) on which the SORTFL is sorted. After creation the SORTFLs and DIRFLs are read-only and shareable. The differential files are denoted by DIFFL and stored within DIFMAN (DIFFerential file MANager). The DIFFLs are very dynamic, so a kind of dynamic hashing is used to store and retrieve tuples by primary key. Indexes on secondary keys are possible. If a DIFFL becomes too large then reorganization is necessary, which means recreating the SORTFLs and DIRFLs of the relation.

Access to VIDEBAS. VIDEBAS is accessed using a derivative of SEQUEL, which offers data manipulation and retrieval. The program SQLMAN (Se-Quel MANager) accepts input and composes the answer with help of DIFMAN and the contents of the read-only part of the database. SQLMAN exploits also a bufferpool to satisfy data requests to SORTFL and DIRFL. An instruction for DIFMAN concerns only one differential file. Retrieval from DIFMAN is triggered by the instruction "read tup". The condition which has to be satisfied, the required attributes and ordering can be given as parameters. Suppose all EMPLOYEES with AGE = 30 have to be retrieved. First a "read tup" for the DIFFL of the EMPLOYEE relation has to be issued with

Fig. 2. Addressing of physical record.

Fig. 3. VIDEBAS architecture.

AGE = 30 as condition and the primary key as sort criterium. Second the right SORTFL will be accessed offering all EMPLOYEE records with AGE = 30, also ordered on primary key. A merge gives the final result. Updates are performed one record at a time.

3. MODELS

3.1 Validation of formulas

Except the formulas concerning external memory occupation, all DBTG formulas are taken from[3]. This UDS model is, to my knowledge, the most validated DBTG model.

The used VIDEBAS formulas are partly borrowed from UDS and partly derived specially for this purpose. The underlying system is however simple, so the validation may not be too difficult.

3.2 UDS-model

In[3] it is assumed that accesses to DBTT and FPA can be satisfied by the bufferpool. An entry in the DBTT contains some physical addresses of four bytes each. As each record has an entry in a DBTT, this table may become rather large. When a record is not stored anymore on its original page, then the DBTT is used to locate the record. This implies that given a POINTER a record is fetched in one page access. Other uses of the DBTT are in FIND with DATA BASE KEY, FIND OWNER and other DML statements. The FPA only contains two bytes per page, so is small.

In this paper record indexes are assumed to be two levels deep. This is sufficient to handle even large files. As on one page a few hundred of entries may be stored POINTER-ARRAYs and set indexes are assumed to occupy only one page.

Other assumptions concern the number of accesses needed to retrieve tuples, to scan a set, to update the database and so on. These matters will be treated in more detail in Section 5.

3.3 VIDEBAS-model

It is assumed that the needed DIRFL pages are always in the bufferpool. The DIRFLs contain one (small) entry per page, (not per record!). The DIRFLs are static and key compaction techniques can be used. Normally there will be more than one DIRFL per relation, say three or four. In total this still means a small charge on internal memory. A SORTFL page is assumed to be never in the bufferpool.

DIFMAN is stored in virtual memory and is assumed to occupy at most a few megabytes. This limits the update rate. Suppose a relation has $10^{**}6$ tuples, each 100 bytes long. Then the "real" data occupies 100 Mb. So 10,000–20,000 *different* tuples may be updated before a reorganization is necessary. This may be not too bad. In[5] a student database of hundreds of megabytes has been described in which each day 50,000 transactions cause read accesses to 10,000 different records, while only 2300 different records were updated. So a strong locality of reference on record level has been observed.

The "hit-rate" is defined as the probability that a page is in the (system) bufferpool when a reference to a DIFFL or to an index of a DIFFL is done. In ([6], p. 106) it is stated that for many database environments hit rates of 0.8 are observed, "aided, substantially by frequent references to indices", like is the case in DIFMAN. Measurements by Tuel[7], Sherman and Brice[8] and Loomis and Allen[9] support this statement and show that hit rates of 0.8 are measured for bufferpools of 20–40 pages and a wide variety of environments. The mentioned measurements concern rather large databases, while the DIFFLs will be small in general. Moreover, at the cost of a small internal memory bit map, a Bloom filter can prevent many random accesses to DIFFL records, see[5]. Such a filter has not been implemented in VIDEBAS until now, however! Summarizing, it is assumed that a data reference in DIFMAN always requires 0.2 page access.

Another model parameter concerns the speed of the paging device. Often access to such a device is much faster than to a normal external medium. Moreover operating systems tend to favour paging activities. This speed ratio is taken into account in the model.

Indexes in DIFMAN are significant smaller than in UDS; also here however two level indexes are assumed.

The number of tuples in DIFMAN which satisfies an equality condition is important for the evaluation. Although a DIFFL is small compared to a SORTFL it is assumed that 10% of the qualifying records in a SORTFL qualify in a condition issued for DIFMAN.

3.4 Mapping of storage structures

According to the following rules a UDS database will "generate" a VIDEBAS database. It is stressed that throughout this paper these "generation rules" will be applied.

—LOCATION MODE IS CALC implies a SORTFL with DIRFL for the hashed attribute.

—RECORD SEARCH KEY causes a SORTFL and DIRFL; the SORTFL ordered on index attribute.

—A SET causes a SORTFL with DIRFL; the SORTFL ordered on the attribute which defines the relationship; it is assumed that all sets are non-information bearing!

—SET SEARCH KEY causes a SORTFL and DIRFL; the SORTFL ordered on set and index attribute.

—The same holds for the clause SORTED INDEXED: a SORTFL with DIRFL is caused. The SORTFL ordered on set and index attribute.

Updates imply in VIDEBAS a DIFFL per relation. As stated before in DIFMAN dynamic hashing is used for the primary key. An index is defined for each secondary key for which a SORTFL with DIRFL exists.

In the sequel some abbreviations concerning a record type are used. Between brackets adopted values are given.

NPAG	number of pages to store all records of a type	external memory occupation we get:
NDUP	number of records with the same attribute value (1, 5, 20)	$NPAG*(1.20 + NRSK*0.15 + NSET*0.08 + NSSK*0.15).$
NRPG	number of records per page (10)	
NRSK	number of RECORD SEARCH KEYs (0, 1, 2)	<i>VIDEBAS system</i>
NSET	number of set types in which is participated, either as owner or as member (0, 1, 2, 3, 4)	In VIDEBAS the space occupied by the DIRFLs is neglected. One SORTFL with DIRFL is needed for the primary key. It follows from the mapping rules, see 3.4, that each index generates one additional SORTFL with DIRFL. If the record participates in a set as owner, then no extra SORTFL with DIRFL is needed, otherwise it is. The record participates in NSET sets types, so this causes an external memory occupation of $NPAG*NSET*0.5$. One SET SEARCH KEY can be taken into account by the SORTFL with DIRFL for the concerning set, see 3.4. It is assumed that never a second SET SEARCH KEY for a certain set type will be necessary. The external memory requirements will be given by the following formula and Table 1 summarizes the results:
NSSK	number of SET SEARCH KEYs per set with this record as member (0, 1)	
NMEM	number of members in a set occurrence (3, 20, 100)	
RAT	ratio of speed of paging device and speed of external medium (1, 4)	

It is assumed that to read one page from external medium takes as much time as to write one on it. This is noted as one page access (1 PA). An access to the paging device requires (1/RAT) PA.

4. EXTERNAL MEMORY OCCUPATION

UDS system

In UDS overflow pages are needed. Moreover most pages will contain some free space. In total 20% extra space is counted.

An index for a record type has entries consisting of two-tuples <attribute value, POINTER>. Such an entry is estimated to be 16 characters and taking into account a normal load factor and record size, of say 140 bytes, an index is assumed to take 15% space extra.

Sets can be implemented in three ways. Also when a set is small, a POINTER-ARRAY requires at least one page. So we assume sets to be implemented as chains as this is more efficient. An owner record needs 8 or 16 bytes extra for POINTERS; a member record, depending on some options between 8 and 19 bytes. When a record participates in a set, either as owner or as member 8% will be counted as overhead for the set connection data.

Set indexes are used when sets are large, so only when a few set occurrences exist. The same estimate as for record indexes is used for this. The space needed for DBTT and FPA is neglected. For the total

$NPAG*(1 + NRSK + NSET*0.5).$

5. DATABASE ACCESSES

In both models it is assumed that only one user is present. In[4] it is shown by measurements that, in an update environment POINTER-ARRAYs allow an average good set implementation. In[10] chaining has not been taken into account as POINTER-ARRAY seemed to offer a good alternative. Only when scanning a set (see 5.1.2), chaining is a little bit faster. So for simplification we do not consider chaining from now on.

In the environments considered the number of records with the same value for the concerned attribute (the number of duplicates), is small compared to the number of pages in the file. This allows the simplicity of the formula DIRECT (see below).

5.1 Retrieval operations

Retrieval operations are necessary to give output in case of a retrieval request. Moreover some retrieval of data may be necessary as a preparation to update

Table 1. Quotient of external memory occupation of UDS and VIDEBAS

		NSET	0	1	2	3	4
NSSK=0							
NRSK							
	0		1.20	0.85	0.68	0.57	0.51
	1		0.67	0.57	0.50	0.45	0.42
	2		0.50	0.45	0.41	0.39	0.37
NSSK=1							
	0		1.35	0.95	0.75	0.64	0.56
	1		0.75	0.64	0.55	0.50	0.46
	2		0.55	0.50	0.45	0.42	0.40

operations. For instance to modify a tuple the tuple has to be read first, see also 5.2.

DBTG systems offer facilities to navigate through a database. Among these are facilities to access records, locate members of a set and find an owner of a set given a member. The first two facilities are considered in more detail below. In UDS a member contains the Data Base Key of the owner and in one page access the owner can be fetched. In VIDEBAS the primary key of the “owner” is stored in the “member”. It will appear that this works very fast too.

Both systems will often use the formula CONSEC which gives the number of page accesses needed to retrieve N consecutively stored records. CONSEC is borrowed, like all UDS formulas from[3]. When $1 < N \leq \text{NRPG}$ then the records may occupy one or two pages. An estimate of the number of accesses is 1.5 PA. CONSEC is defined as:

$$\begin{aligned} &= 1 \quad \text{when } N = 1 \\ \text{CONSEC}(N) &= 1.5 \quad \text{when } 1 < N \leq \text{NRPG} \\ &= N/\text{NRPG} \text{ otherwise.} \end{aligned}$$

5.1.1 *One recordtype*. The following query will be treated: “Fetch records of type A which satisfy $\text{att}(A) = \text{constant}$ ”. Here “att(A)” is an attribute of A. According to its definition, NDUP tuples will satisfy the condition.

UDS system. The concerned attribute “att(A)” may be hashed or indexed. LOCATION MODE IS CALC implies hashing. It costs 1.33 PA to fetch the first record with a certain key value and 0.33 page access for each of the next duplicates, see[3]. The time needed to retrieve NDUP records is $(1.33 + (\text{NDUP} - 1) \cdot 0.33)$ PA.

An index on record level is forced by a RECORD SEARCH KEY clause. The root page will often be in the bufferpool, so the mean access to the root page will be estimated by 0.25 PA. An index is two levels deep and the access to the second level requires one PA. Access to the second level of the index gives the addresses of the records which satisfy the equality condition. The number of duplicates is small compared to the number of pages in a file, so the formula DIRECT which gives the number of accesses given the addresses is simple.

$$\text{DIRECT}(N) = N.$$

VIDEBAS system. First the DIFFL has to be

accessed. If $\text{NDUP} = 1$ then it is assumed that the retrieval concerns the primary key. Via dynamic hashing the page on which the tuple may reside is read (0.2 PA). An updated tuple is found in DIFMAN. In that case VIDEBAS does not have to access a SORTFL, which means that a fast access is achieved. This possibility is neglected.

If $\text{NDUP} > 1$ then the number of duplicates in DIFMAN is estimated by $\text{ROUND}(0.1 \cdot \text{NDUP})$, see Section 3.3. To retrieve these duplicates two index accesses are needed extra, so in total we get $(\text{ROUND}(0.1 \cdot \text{NDUP}) + 2) \cdot 0.2/\text{RAT}$ PA.

Now the SORTFLs with DIRFLs still have to be accessed. The pages of a DIRFL are supposed to be in the bufferpool, so retrieving NDUP tuples requires $\text{CONSEC}(\text{NDUP})$ PA.

The results are given in Table 2.

5.1.2 *One to many relationship*. The query under consideration can be formulated as: “For each A record satisfying $\text{att}(A) = \text{constant } 1$, fetch the B-records related to the A-record and satisfying $\text{att}(B) = \text{constant } 2$ ”. For each A-tuple retrieved some B-tuples have to be fetched. The retrieval of A-tuples has been treated in 5.1.1, so we concentrate on the fetching of the B-tuples. In UDS this strongly depends on the implementation of the set with A as owner and B as member.

UDS system. Four cases will be treated. In the first case a set index is present and the members of the set are physically ordered on indexed attribute. In UDS this is realised by the clauses MODE IS LIST and ORDER IS SORTED INDEXED. The physical address of the index, which is only one level deep, is stored in the DBTT, so in a buffer. The number of page accesses is given by:

$$1 + \text{CONSEC}(\text{NDUP}).$$

Next, a set index is present and no usable physical ordering. This is caused for instance by MODE IS POINTER-ARRAY, ORDER IS IMMATERIAL and SEARCH KEY for att(B). Again the physical address of the index is in a buffer, the number of records which satisfy the condition is NDUP, so the following formula characterizes this situation:

$$1 + \text{DIRECT}(\text{NDUP}).$$

The third case is characterised by physical ordering on set membership and no set index: MODE IS LIST and ORDER IS IMMATERIAL. The DBTT entry which gives the physical address of the first member,

Table 2. Number of page accesses for retrieval of one record type. (RSK means RECORD SEARCH KEY)

NDUP	UDS		VIDEBAS	
	CALC	RSK	RAT=1	RAT=4
1	1.33	2.25	1.2	1.05
5	2.65	6.25	2.1	1.65
20	7.60	21.25	2.8	2.2

is in the buffer, so the number of accesses is:

CONSEC(NMEM/2) if NDUP = 1

CONSEC(NMEM) if NDUP > 1

Finally, no index and no physical ordering, which is realised by **MODE IS POINTER-ARRAY** and **ORDER IS IMMATERIAL**. If NDUP = 1 then on the average half the set, otherwise the whole set has to be scanned.

SEQNT(NDUP, NMEM) = NMEM/2 if NDUP = 1

= NMEM if NDUP > 1.

VIDEBAS system. In VIDEBAS this query amounts to an equijoin operation. To treat this, each relational data base system invokes an optimizer. We assume that the optimizer first fetches the A-tuples and then for each A-tuple the connected B-tuples. It is the last operation in which we are interested. Four cases are treated within UDS of which the first two cases consider an index. According to our assumptions, see section 3.4, a set with an index implies a SORTFL and DIRFL with ordering on set and index attribute. The third and fourth case concern an UDS set only. This implies a SORTFL and DIRFL with ordering on the attribute which defines the relationship. In both cases the needed DIRFL pages are supposed to be in the bufferpool. Now, basically the formulas of 5.1.1 can be used again to compute the number of accesses. Tables 3 and 4 give the results for the index and non-index case respectively.

5.2 Updates

The costs connected to updating can be divided into several parts, see also[11]:
the cost to reading the page to be updated,
the cost of rewriting a modified page when selected for replacement by the buffer manager,

- the cost of writing before and/or after image records,
- the cost of backing out transactions,
- the cost connected to end-of-transaction processing,
- the cost of checkpointing the database state,
- the cost to restart after a system down, when all data in volatile memory has been destroyed and
- the cost to recover from a medium failure (for instance a head crash).

The latter two costs are not considered as they have to be paid only a few times a week and a few times per year, respectively. However, it is important of course that the time to restart or to recover stays within reasonable bounds.

In UDS the needed DBTT and FPA pages are assumed to be in the bufferpool. When those pages have been modified rewriting will be neglected. It is assumed that the before and after image files are written on a record basis keeping the number of page accesses low. Backing out is fast as the before image file is small. At end-of-transaction all modified pages are written to the database, thus preventing check-point operations. In[11] it has been shown that in environments with a not too high update frequency this approach is perhaps not optimal, but at least reasonable well.

In VIDEBAS the before image file is kept in virtual memory and backing out is fast. Only after image records are written to a log. Checkpointing is necessary as updates are applied to DIFFLs, which are stored in volatile memory.

Summarizing, two aspects will be considered: the update part with the end-of-transaction handling and checkpointing.

5.2.1 *Update cost.* If a tuple has to be modified then it has to be brought into internal memory first. This “preparation” implies a retrieval operation. In this section it is assumed that all preparations to the

Table 3. Number of page accesses to retrieve from a one-to-many relationship (SI means SORTED INDEXED, SSK is SET SEARCH KEY)

NDUP	UDS		VIDEBAS	
	LIST,SI	SSK	RAT=1	RAT=4
1	2	2	1.2	1.05
5	2.5	6	2.1	1.65
20	3	21	2.8	2.2

Table 4. Number of page accesses to retrieve from a one-to-many relationship (non-index case)

NMEM NDUP		UDS		VIDEBAS	
		LIST	SEQ	RAT=1	RAT=4
3	1	1.5	1.5	1.9	1.6
20	1	1.5	10	2.1	1.65
20	5	2	20	2.8	2.2
150	1	7.5	75	9.40	7.95
150	5	15	150	18.4	15.85
150	20	15	150	18.4	15.85

update operations has been performed already, as those preparations concern retrieval operations which have been treated in the previous section.

For UDS this means that, if needed currency indicators are set and current-of-run-unit (CRU) and current-of-set (CoS) are already read into the bufferpool.

In VIDEBAS no presence checks are taken into account and a tuple which has to be modified has already been brought into internal memory.

UDS system. When LOCATION MODE IS CALC then storage of a tuple requires the reading of a page (1.33 PA) and the rewriting of it (1 PA). If the record type is member of a set type with MODE IS LIST and ORDER IS SORTED INDEXED then the access to the index page precedes the reading and rewriting of the data page. In total this makes 3 PA. In all other cases only 2 PA are needed.

If a one-to-many relationship has been established then the "member" has to be updated (the Database Key of the owner has to be set in the member) and the POINTER-ARRAY which defines the set also. As the CRU is in the bufferpool only rewriting of the CRU has to be counted. Moreover reading and rewriting of the page of the POINTER-ARRAY. In total 3 PA. The modification of the CRU only implies the rewriting: 1 PA. The same holds for the erasure of a tuple.

The storage and erasure of records have "side-effects". These concern among others connection and disconnection of members from sets. Only small differences with the explicit actions described above, exist. Another "side-effect" may be the insertion (or deletion) of an entry in a record index. The access of the root page requires 0.25 PA. The second level page has to be read and written making a total of 2.25 PA. A set index is stored on one page, so the insertion of an entry in a set index requires 2 PA.

VIDEBAS system. The storage of a tuple requires an instruction to DIFMAN. Via the dynamic hashing one access to read and one to write the correct page is required, making in total $2 \cdot 0.2 \text{ PA} = 0.4 \text{ PA}$. The updating of an index requires one access to the root page and a read and write access to the second level page. In total 0.6 PA. If a one-to-many relationship has been established then an access to the root page

is necessary, the reading and writing of the second level page and the DIFFL record itself have to be performed. This results in $5 \cdot 0.2 \text{ PA} = 1 \text{ PA}$. $\text{RAT} = 4$ strongly reduces update time. Table 5 gives the comparison between UDS and VIDEBAS.

5.2.2 Checkpointing cost. UDS system. No checkpointing costs are needed here as after each transaction all pages are written to non-volatile storage.

VIDEBAS system. From time to time, a checkpoint of DIFMAN has to be taken. This implies the dumping of the core image to non-volatile storage. Assuming blocks with the length of one track the reading and writing of 1 Mb requires about 70 (long) accesses.

6. FINAL REMARKS

Read-only part

The read-only part of VIDEBAS now only contains SORTFLs and DIRFLs. It is easy to expand this to HSAM files, hashed files, and so on. The price for it may be a more complicated reorganization. A related point is that in UDS each addition of an index slows down the update operations. This hardly holds for VIDEBAS, see Table 4. So it can be advantageous to add a SORTFL in VIDEBAS whereas it does not pay to add an index in UDS.

Reorganization

In VIDEBAS reorganization can be executed per relation. As soon as the DIFFL becomes too large new SORTFLs with corresponding DIRFLs have to be made. A reorganization is more or less a merge of the "old" SORTFL with DIRFL and the DIFFL to create a "new" SORTFL and DIRFL.

If a relation occupies 100 Mb and if three SORTFLs are present, then 300 Mb are necessary to store this relation. Counting 50 ms to access a page of 3000 bytes, then the reorganization takes about 10,000 sec. In a DEC-system10 it is possible to claim contiguous space for a file. Doing this for SORTFLs and DIRFLs and anticipating on it, the reorganization can work with large buffers of say 30 Kb. This would substantially decrease the reorganization time.

In UDS much less reorganizations will be needed. Per reorganization it will probably take more time.

Table 5. Number of page accesses to update the database. (SI means SORTED INDEXED)

		UDS	VIDEBAS	
			RAT=1	RAT=4
	CALC	2.33		
Storage of a tuple	NOCALC	2	0.4	0.10
	LIST SI	3		
Connect		3	1.0	0.25
Disconnect		3	1.0	0.25
Modify		1	0.4	0.10
Erase		1	0.4	0.10
Insertion in record index		2.25	0.6	0.15
Insertion in set index		2	0.6	0.15

Which of the two effects will dominate depends strongly on the environment.

High update environment

In an environment with many transactions per second, while those transactions update many different tuples, on-line reorganization has to be taken into account. This may cause many additional accesses, specially when the database is large. How VIDEBAS will compare then with a system like UDS or with a system with a very large database buffer, has to be subject of further study.

Internal Memory occupation

UDS consists of one program which looks after all contacts with the database. In this program one bufferpool is present. To make an optimal use of the devices on which the data base resides UDS uses internally multi-tasking.

VIDEBAS consists of two programs, SQLMAN and DIFMAN, one containing an explicit bufferpool, the other (DIFMAN) is implemented in virtual memory, so utilizes an implicit system bufferpool. The program code is split up over SQLMAN and DIFMAN. According to the considerations of 3.1 and 3.2 the VIDEBAS bufferpools may be smaller than the bufferpool of UDS. DBTG systems allow rather complex data structures and DML statements. The corresponding VIDEBAS facilities seem to be rather simple. It is rather difficult to judge which of the two systems uses the most internal memory.

7. CONCLUSIONS

In this paper a DBTG system (UDS) has been compared with a relational system (VIDEBAS) in an operational environment. VIDEBAS tries to obtain speed at the cost of extra external memory. The ratio between external memory used by UDS and VIDEBAS ranges from 0.37 to 1.35. Perhaps 0.40–0.80 are good averages.

Retrieval is handled better within VIDEBAS: from about a factor 1 to a factor 10. When LOCATION MODE IS CALC and NO DUPLICATES ARE ALLOWED the two systems perform about equal.

Also the set implementation LIST with ORDER IS SORTED INDEXED generates good UDS figures. It is clear however that these clauses can be applied only once per record type. In all other cases VIDEBAS performs significantly better.

Update operations are performed much faster in VIDEBAS than in UDS. Always a factor 2–6 is achieved (for a paging device with a speed equal to the speed of the external medium) and a factor 8–24 if the speed ratio is four. In VIDEBAS, however, checkpointing is necessary. This takes from time to time several seconds to keep the restart costs within reasonable bounds. The specific performance differences depend, of course strongly on the environment. When only one access path to the records of a certain type is used than UDS will do well. The more paths are used frequently, the better VIDEBAS will perform.

REFERENCES

1. H. M. Blanken: The architecture of VIDEBAS, a relational database management system. *Proc. 6th ACM European Reg. Conf.*, pp. 213–223 (1981).
2. H. M. Blanken: Another approach to DBMS implementation. *Workshop on Relational DBMS*, Toulouse (Feb. 1983).
3. H. Kinzinger and W. Koser: Entwicklung eines SYSTEMS zur optimierung der speicherungsstrukturen in CODASYL databanken. Diplomarbeit, Technische Hochschule, Darmstadt (Aug. 1981).
4. W. Effelsberg *et al.*: Measurement and evaluation of techniques for implementing Cosets- a case study. *Proc. Int. Conf. on Data Bases*, pp. 135–160 (July 1980).
5. L. L. Gremillion: Designing a bloom filter for differential file access. *Commun. ACM* (Sept. 1982).
6. G. A. Champine: Four approaches to a database computer. *DATAMATION*, pp. 101–106 (Dec. 1978).
7. W. G. Tuel Jr.: An analysis of buffer paging in virtual storage systems. *IBM J. Res. Development* **20** (1976).
8. S. W. Sherman and R. S. Brice: Performance of a database manager in a virtual memory system. *ACM Trans. on Database Systems* (1976).
9. M. E. S. Loomis and F. W. Allen: Paging behavior and performance optimization in a CODASYL DBMS. *Proc. Int. Conf. on Data Bases*, pp. 119–134 (July 1980).
10. H. M. Blanken: Automatic generation of storage structures for a DBTG database system. *Proc. Int. Conf. on Data Bases*, pp. 99–119 (July 1980).
11. A. Reuter: Performance analysis of recovery techniques. Internal Rep. 63/82, Univ. of Kaiserslautern (1982).