Monitoring database system performance

P. Dearnley

School of Computing Studies, University of East Anglia, Norwich, Norfolk

The motivations and methods of monitoring data base system performance are discussed. The techniques used to monitor a particular system are described and the results of monitoring three periods of data base system operation are given.
(Received November 1976)

New methods of managing data and new data base management software packages are continuing to appear. Both data base system designers and users, or potential users, are interested in evaluating new methods and products. One method of evaluation is to monitor the performance of a system whilst processing different query loads. The monitoring information obtained may be used by users to compare one system with another or to assess the implications of moving to a data base management system from a conventional filing system. The interests of system designers are a little different; they are interested in evaluating performance to improve their product and in not only the overheads in data base systems in general, but the overheads attributable to their particular data base management philosophy.

The data base system designer is unlikely to be able to perform an analytical evaluation of his entire product due to both the complexity of the code employed and the method of operation of the system, although it is possible to perform such evaluations for some part of the design process (Kollias, 1976). The operation of a system such as IMS (IBM, 1970) or a CODASYL implementation (CODASYL, 1971), allows the programmer considerable freedom to navigate through the data base (Bachman, 1973); an analytical approach would be unable to allow for the vagarities of such navigators. Similarly the implementation of non-navigational systems often involves the dynamic selection of access paths based on relationships in the data base at run time (Hutt, 1976; Dearnley, 1976). Hence any evaluation is likely to be in the form of experiments on a running system or on a model of a system. If the analysis of the results can be tied to the logical structure of the system then the designer can decide how to allocate resources to improve the system. The resources may be in the form of additional or special hardware (Lin *et al.*, 1975; Coulouris *et al.*, 1972) to speed the performance of a given system. Alternatively the designer may wish to know where to concentrate human resources on the production of new algorithms, firmware, assembly language code, etc. for an improved software package. Similarly the costs of running the system can be divided between useful work, overheads inherent in the data base approach and the overheads peculiar to the particular data base management philosophy employed.

Methods

One simple method of performance analysis is to regard the data base management system as a 'black box' and to measure the computing resources consumed during the processing of particular query loads. This method may be sufficient for users performing comparative analyses prior to selecting a particular product but since it is not tied to the logical structure of the system it gives the designer little insight into the operation of his product. A slightly more sophisticated approach is to sample and record the state of the computer at regular intervals. This can be thought of as *physical monitoring* and can give some insight into the usage of machine resources. In addition if the software is produced by a system generation procedure which produces a loader map it may be possible to work back through this map and deduce the proportion of time spent in those routines which happen to be sampled. This method may be misleading if the system is overlaid and it does not give information about the frequency of use of routines. Hence although the method may be useful for tuning computer performance it also contributes little to understanding the system. *Logical monitoring* can be employed if the source text of the system is available. Calls on monitoring procedures are inserted in the source text. The monitoring procedures record the entry to and exit from routines or groups of routines. The information recorded by the monitoring routines is output at the end of each monitored run. The information might include the processor time used, the real time used, the frequency of use and the number of peripheral transfers performed. The timings may be made on a *cumulative monitor* basis so that the time attributed to a routine includes all lower level routines called by the given routine, or, on a *differential monitor* basis so that the time used by lower level routines is excluded. Logical monitoring ties the evaluation to the structure of the system, can accommodate overlays and allows measurements other than just the state of the host computer.

Application

The method of logical monitoring has been used on a self organising data management system (Stocker and Dearnley, 1973, 1974a). To investigate the viability of such systems a model system has been built (Dearnley, 1974a) and a number of experiments performed (Dearnley, 1974b, 1976). The designers wished to know both the overheads attributable to this type of system in general and the areas in which to concentrate in the design and implementation of a new full-scale system following the same philosophy. The new system is being implemented on a microprogrammable 'back-end' computer (Canaday, 1974) and the designers have to choose between firmware, assembly language and high level language for the implementation of various routines. Naturally the monitoring does not produce absolute guide lines for implementation nor does it produce perfect measurement of overheads but it does give some quantitive information to assist in these difficult areas.

Each major routine is equipped with calls on monitoring procedures. These procedures record the real time used and the processor time used on both a cumulative and a differential basis. In addition the frequency of use is measured, as are the transfers to and from the disc unit holding the data base. Peripheral transfers involving files in the operating system file-store but outside the data base are ignored. When the data base system is closed down the monitoring information is written to file-store for subsequent analysis.

Downloaded from https://academic.oup.com/comjnl/article/21/1/15/445424 by guest on 19 April 2024

Structure \ Utility	*Housekeeping*	*Processing*	*DBMS overhead*	*SODMS overhead*
High level		Searches	Interpreters	Extract Sort Index Route finding Route costing Folio management
Low level		Logical I/O Page turning Basic IOCS Move Compare	File allocation Version definition I/O	Folio definition I/O Folio definition I/O
Miscellaneous	Archiving Start up	I/O thru' OS		Folio definition Statistics

Fig. 1 Classification by structure and utility

System structure

To appreciate the results of the studies presented in later sections it is necessary to understand the logical structure of the system under evaluation. The system is controlled by two interpreters. One interpreter allows the user to specify two types of actions, classified as simple or complex. Simple actions include such things as loading new user input and writing results to file-store. Complex actions call upon a route finding and costing section which prepares a route through the data base to answer an enquiry. The route finder, in turn, calls upon a second interpreter to perform complex actions such as searching and re-structuring. This second interpreter can also be invoked by the folio management routine which takes decisions about restructuring and updating when the system is idle. All action routines call on a common set of file handling routines which manage both the data base itself and the system's directory of folio and version definitions. The file handling routines are themselves monitored by statistics routines which record the usage of versions whilst searches are in progress. A number of miscellaneous routines exist for such purposes as system start-up and archiving. More detail of the implementation is given elsewhere (Dearnley, 1974a).

Classification by structure

For monitoring purposes the system is divided into 57 routines. These routines are, for the purpose of brevity in this section, grouped into 21 categories. The categories are shown in the body of **Fig. 1.** The rows of Fig. 1 show the routine categories further grouped into three classes: high level, low level and miscellaneous.

The high level class includes those routines of which the user is aware and those routines which represent major components of the system. Thus the class includes the user interpreter and the costing mechanism, both of which are known to the user. It also includes re-structuring, route finding and folio management which represent major parts of the system.

The low level class includes minor routines necessary for system operation and those routines from which the major components are built. Thus this level includes folio and version definition handling, the input/output package, and, the move and compare routines. The input/output package comprises record handling at the logical level which calls upon page turning routines as required. The page turning routines process the system map and call upon the basic IOCS for physical transfers.

The miscellaneous class comprises the necessary routines for archiving and recovering the data base, starting the system up, communicating with the host operating system file-store, collecting statistics and receiving definitions of new data from the user. The statistics group is included here rather than under low level because it operates both at the search level (to collect the nature of the query) and at the IOCS level (to record how the query is serviced).

Classification by utility

To provide data on the overheads of the data base approach and the self organising philosophy the 21 categories are divided into four classes based upon their utility from the user's viewpoint. The classes are housekeeping, processing, DBMS overhead and SODMS overhead. The classification is shown by the columns of Fig. 1. The processing class includes those groups which represent the work a user actually wishes to do and might explicitly program in a conventional file processing environment. Thus it includes putting data in and getting it out (I/O through OS), searching for records matching query specifications and extracting relevant data (searches with their necessary I/O, moves and compares). Loading and initialising the system and keeping back up is classed as housekeeping. The division between DBMS and SODMS is more difficult, for example where should (re-)structuring be placed? One view is that it is a necessary prerequisite of using the DBMS and therefore not in the SODMS class. However the approach taken here (to the advantage of the DBMS class) is that structuring and re-structuring is performed dynamically to achieve cost advantages and thus should be charged to the SODMS class. Thus the DBMS class comprises interpreters, (dynamic) file allocation and version (i.e. data file) definition I/O; rather a generous view of DBMS overheads! The SODMS class includes statistical routines, folio management, route finding and costing, all structuring and obtaining folio definitions and the I/O of folio definitions.

The implication of using the term 'utility' is that the user should view the processing class and, to some extent the housekeeping class, as useful in obtaining his objectives. Whilst the DBMS and SODMS classes are the overheads in adopting the data base approach in general and the self-organising approach in particular.

Studies

Three studies have been used for evaluation; the studies were

Downloaded from https://academic.oup.com/comjnl/article/21/1/15/445424 by guest on 19 April 2024

deliberately chosen to include archiving, re-structuring and a number of searches. The data base system was run on an ICL 1903T under the GEORGE 3 operating system; it was allocated 30K 24-bit words of store and the data base, system map and system directory were held on one EDS 60 disc transport. To facilitate timing no other jobs were run by the operating system during the monitoring periods. Before the monitor output was analysed a pair of 'dummy' systems were run. One dummy consisted of performing a large number of 'opens' and recording the times taken by the operating system to connect the program to its files. An average overhead for opening a file under this operating system was then subtracted from the monitor output. The second 'dummy' consisted of running the monitoring procedures, in a null routine, a large number of times; this allowed the overhead of monitoring to be assessed.

The monitor output includes the number of peripheral transfers, the cumulative and differential processor times, the cumulative and differential real times and the frequency of usage for each of the 57 routines. For the purposes of this paper the 21 groupings mentioned above are further contracted to ten groups as shown in **Fig. 2,** and the results presented are limited to the frequencies of usage and differential times.

The monitor output from these studies is given in **Figs. 3, 4** and **5.** (Readings which were non-zero but too small to show on the bar charts have been rounded up to a single asterisk; very long bars have been abbreviated and their actual value recorded below the appropriate bar.)

The studies all use one folio of 60 pages of data located in a data base area of 2,400 pages.

Routine Group Name in Fig. 1	*Group for Figs.* 3-5	*Abbreviation used in Figs.* 3-5
Extract Sort Index	Organisation	ORGANISE
Search	Search	SEARCH
Interpreters	Interpreters	INTPRTR
Route finding Routse costing	Route handling	ROUTES
Folio management	Folio management	FOLIO
System directory I/O File allocation Logical I/O Page turning Basic IOCS	I/O	I/O
Move Compare	Move/Compare	MOVE/COMP
Archiving	Archiving	ARCHIVE
Start-up Folio definition I/O thru' O.S.	Other	OTHER
Statistics	Statistics	STAT

Fig. 2 Re-grouping and abbreviations

Study 1. Access and Re-organisation

The first study involves both requests to be answered and an occurrence of data base re-organisation. A serial search is performed which takes the statistics for the usage of that part of the folio over the given threshold. The system then goes idle and into the folio management routines; these routines find that creating a new version would appear to be profitable. The complex action interpreter is instructed to extract a new version containing five frequently used fields, to sort this version on the most frequent access key and to build an index to the version. A further request is then processed. The route for answering this request involves three indexed searches, one of which happens to use the newly created version. Inspection of Figs. 3, 4 and 5 shows that the process category is the most frequently used and that this category uses the most processor and real time. The SODMS category uses roughly half the processor and real time of the process category for about one quarter the frequency of usage; this is partly due to the fact that the route finding and folio management routines are comparatively complex relative to the search and I/O routines. The monitor output for the breakdown by utility shows a considerable amount of the real time in organisation and in I/O, as might be expected in an example involving sorting. The small values given for DBMS must be viewed in the light of the fact that the breakdown into DBMS and SODMS overheads favours the DBMS category by placing all organisation in the SODMS category.

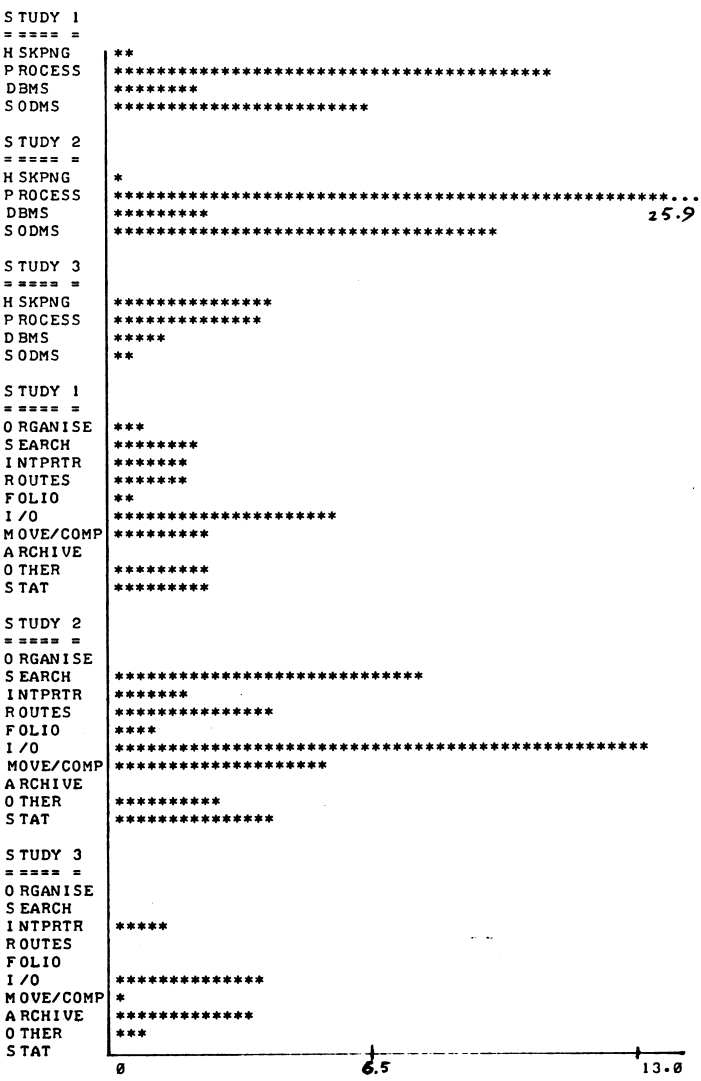

Fig. 3

Downloaded from https://academic.oup.com/comjnl/article/21/1/15/445424 by guest on 19 April 2024

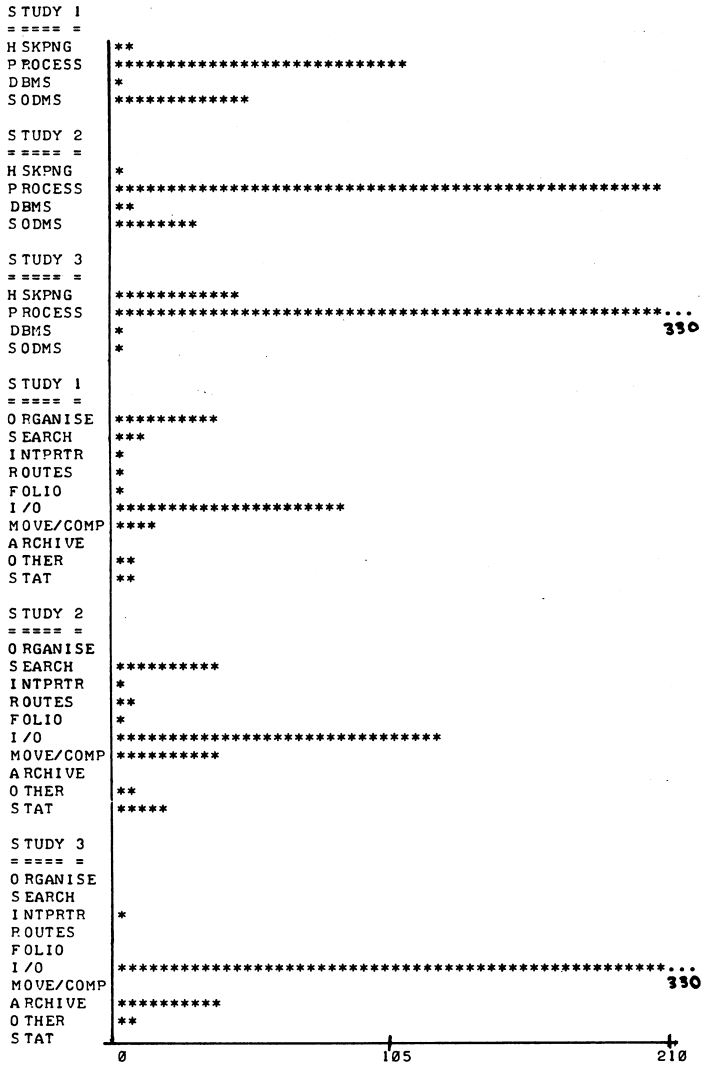

Fig. 4

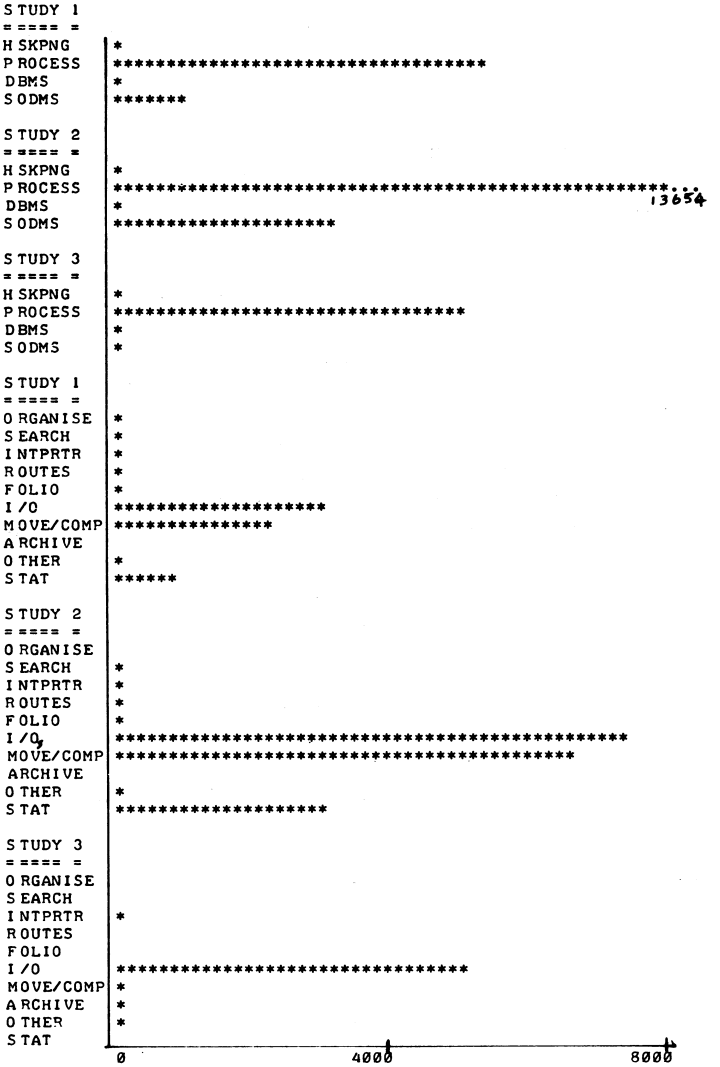

Fig. 5

Study 2. Access only

The second study consisted of a series of six requests to be answered from the data base already online. Each request comprised checking the definition of a folio, specifying selection and output parts of a query, agreeing to the cost of access and returning the results to the host operating system. When the system closed down no re-organisation was found to be profitable. No archiving was involved in this study. Thus this study represents the data base in a steady state. The requests were answered with ten searches, seven of which used indexes and three of which were serial searches. The figures show that major cost is attributable to the process category and that this exceeds the SODMS overhead by a factor of at least 2·5; similarly the SODMS overhead exceeds the DBMS overhead by a factor of at least three. A large part of the process cost is, as might be expected, in the I/O and move/compare routines, whilst the collection of statistics proves to be a significant part of the SODMS category. The processor time for route finding shows that the routes were discovered in just over half a second each.

Study 3. Restoring a dumped data base

The final study involves checking the specification of the data base currently loaded, reloading an old copy of the data base from a physical dump and checking the specification of this copy. This task was undertaken to allow a series of experiments to be repeated from an identical initial data base. The study is included to show the residual overhead when performing a task which requires almost none of the SODMS or DBMS features but is performed using the data base system. The figures show this overhead to be very small in real time with the I/O group using most of the time. The processor times are less satisfactory with almost as much time being used in the control of archiving as in the transfer of data.

Overall assessment

Fig. 6 shows the total times and the peripheral transfers for each study. The ratio of processor time to real time shows that, in this system, comparatively little processor time is used; this corresponds to verbal observations made by users of certain CODASYL-based systems but is in marked contrast to observations of the implementors of, at least, one relational system. Unfortunately such figures are rarely published and thus accurate comparison is difficult. With the exception of study three, which approaches an ideal of 0·07*, the average time per data base transfer is disappointing. It is assumed that this is accounted for by other peripheral transfers to and from the operating system, and, the offline printing of results. The total volume printed during the monitoring (results, data base system messages and operating system messages) was approximately 3,900 lines. Apart from the overhead of opening a file, known to be significant, no allowance has been made for operating system overheads.

*Computed as (52 msec. for average aim movement of 200 cylinders) + (12·5 msec. latency) + (0·5 msec. transfer time) + (25 msec. allowance for read-after-write check on 20% of transfers, i.e. 5 msec.) = 70 msec.

Downloaded from https://academic.oup.com/comjnl/article/21/1/15/445424 by guest on 19 April 2024

Conclusions

In the studies monitored the times attributed to DBMS and SODMS overheads were always less than those for 'useful processing'; in some cases the overheads were appreciably smaller. Until figures are published by other implementors, it is not possible to compare the self organising philosophy with, say, the network philosophy but it is apparent that certain overheads, such as the collection of usage statistics, are quite considerable.

When considering the full-scale implementation of a self organising system it would appear that special hardware, such as head-per-track devices or the LEECH processor (McGregor *et al.*, 1976) would be advantageous in reducing I/O times. This would place more emphasis on the routines which must be completed before the next I/O operation can be performed, for example, the move and compare group. The machine used for the monitored implementation already has special hardware for data movement and the non-numeric comparison routines were coded in assembly language; despite this the processor time used in the move and compare group is considerable. Often the I/O routines have to wait whilst a page of data is examined for matching keys and the appropriate records moved out to another buffer. Alternate buffering helps in sequential searches but many of the tasks require random access, for example indexed searches and access to the system directory, thus the time spent comparing and moving is important and these routines are candidates for implementation in firmware. Similarly it is hoped that some of the control of basic I/O, for example the merging within sorting, can be microprogrammed, further reducing the time elapsing between I/O operations.

	Study 1	2	3
1. Total real time (secs)	179	249	382
2. Total processor time (secs)	19	37	9
3. Ratio 2 : 1 (approx.)	1 : 9·6	1 : 6·7	1 : 44
4. Net real time 2·1 (secs)†	160	212	373
5. Peripheral transfers to/from data base	805	447	5062
6. Average time per data base transfer (4 ÷ 5)	0·199	0·47	0·74
7. Other peripheral transfers to/from OS file store**	1474	1169	651

Fig. 6 Total and transfers
†when monitored the system did not allow overlap.
**obtained from operating system log.

References

BACHMAN, C. W. (1973). The Programmer as Navigator, *CACM*, Vol. 16, No. 11.

CANADAY, R. H., HARRISON, R. D., IVIE, E. L. and RYDER, J. L. (1974). A Back-end Computer for Data Base Management, *CACM*, Vol. 17, No. 10.

CODASYL (1971). Data Base Task Group, April 1971 Report.

COULOURIS, G. F., EVANS, J. M. and MITCHELL, R. W. (1973). Towards Content-addressing in Data Bases, *The Computer Journal*, Vol. 15, No. 2.

DEARNLEY, P. A. (1974a). A Model of a Self-organising Data Management System, *The Computer Journal*, Vol. 17, No. 1.

DEARNLEY, P. A. (1974b). The Operation of a Model Self-organising Data Management System, *The Computer Journal*, Vol. 17, No. 3.

DEARNLEY, P. A. (1976). An Investigation into Database Resilience, *The Computer Journal*, Vol. 19, No. 2.

DEARNLEY, P. A. (1976). Dynamic Access Path Selection, Proceedings of the Symposium on Implementing Relational Data Base Systems, University of Southampton.

HUTT, A. T. F. (1976). A Software Architecture for a Relational Data Base Management System, Proceedings of the Symposium on Implementing Relational Data Base Systems, University of Southampton.

IBM (1970). Information Management System/360 Version 2, general information manual, IBM Form No. GH20-0765.

KOLLIAS, J. G. (1976). The Design of Data Base Management Systems using Linear Programming Techniques, Ph.D Thesis, University of East Anglia, Norwich, England.

LIN, C. S., SMITH, D. C. P. and SMITH, J. M. (1975). The Design of a Rotating Associative Memory for Relational Data Base Applications, Computer Science Department Report, University of Utah, Salt Lake City, Utah 84112, USA.

MCGREGOR, D. P., THOMSON, R. G. and DAWSON, W. N. (1976). *High Performance Hardware for Database Systems in Systems for Large Data Bases*, P. C. Lockemann and E. J. Neubold (editors), North Holland Pub. Co.

STOCKER, P. M. and DEARNLEY, P. A. (1973). Self-Organising Data Management Systems, *The Computer Journal*. Vol. 16, No. 3.

STOCKER, P. M. and DEARNLEY, P. A. (1974). A Self-Organising Data Management System in *Data Base Management* edited by Klimbe and Koffeman, North Holland.

Downloaded from https://academic.oup.com/comjnl/article/21/1/15/445424 by guest on 19 April 2024