Database integrity

R. A. Davenport

*Scientific Control Systems Ltd, Sanderson House, 49-57 Berners Street, London W1P 4AQ**

This paper discusses the options available to a systems designer in producing a system that has requirements for high data integrity. Such requirements are often typical of transaction-oriented systems. Models are provided which will allow performance predictions to be made for particular examples.

(Received June 1975)

Data base integrity

System availability may be defined as functional integrity, the ensuring of integrity of the information processing system and of the processes operating within. Data integrity on the other hand is the protection of vital data against loss or damage caused by hardware or software failure. Data integrity therefore may be preserved while functional integrity is not. A system may tolerate a reduced availability but it is unlikely that it will tolerate a loss of data integrity. The most important data is normally that held in the system files, therefore the most important form of data integrity is file integrity, particularly if the files are large integrated data bases dealing with all aspects of an organisation's business. This paper discusses both system availability and file (or data base) integrity. It does not attempt to cover the wider subject of computer security of which availability and integrity are but a part. Computer security in general is the subject of a great deal of ongoing research (Browne, 1972; Canning, 1971; Canning, 1972; Hoffman, 1969; Martin, 1974).

The data base in any information processing system (batch or online) is of the utmost importance containing as it does the information that is being interrogated and/or updated by the terminals or peripherals. Because of its importance, maintaining the integrity of the information held in the data base is of prime concern to the systems designer.

The integrity of the data base may be affected by a software, a hardware, or operator malfunction. The malfunction may be of a magnitude serious enough to either completely destroy the data base or to leave it in a state that its contents cannot be vouched for. It is therefore necessary for the system to possess the ability to recover from such a malfunction while maintaining the completeness (or integrity) of the information held within the data base.

There are a number of malfunctions (error situations) that can arise which affect data base integrity. The errors may be undetectable or detectable. If undectable errors are not trivial the integrity of the data base is already lost. The responsibility rests on the application to devise and execute appropriate consistency checks to reduce the possibility of such errors to as low a figure as possible. Detectable errors may be detected by hardware, by software or manually. Hardware detected errors are either self-correcting which requires no software action, of a type that produces a warning, or instant, which produces an error on failure of the hardware. Software detected errors may either cause an abort of an application program, an abort of the operating system or be errors in the content of the data base.

As well as software and hardware malfunctions, for transaction oriented systems there are the problems of concurrent processing, i.e. the simultaneous accessing of the files by a number of users. The problems as outlined by Waghorn (1968) and Collmeyer (1971) are:

1. *Concurrent update*
This can lead to inconsistencies in the data held by the file and is usually solved by 'locking' records that are to be updated.

2. *Deadlock*
This can be as a result of the locking mechanism introduced to deal with the first problem. Dealing with it may be the responsibility of the data base management system, the transaction monitor or the operating system. Shemer and Collmeyer (1972) have shown that the problem does not occur to a significant extent in a typical system.

3. *Time consistency of data*
This can be due to chains of insertions and deletions eventually causing inconsistencies to develop, due to the order in which they are applied, as discussed by Waghorn (1968). Florentin (1974) has proposed the use of mathematical logic as an aid to dealing with this problem.

Methods of preserving integrity
There are three distinct methods for preserving data base integrity.

1. *Generation*
In this method (often known as the grandfather/father/son method) transactions are combined with one physical version of a data base to produce a new physically separate version. The previous version and the transactions that accessed it are maintained as well as the new version. In the event of a malfunction the file is recreated from the old version and the transactions. More than one generation of the data base and its relevant transactions may be maintained.

2. *Dumping*
If a data base is updated by the updateinplace method where the old version of a record is overwritten by the new version then the generation method cannot be applied. An alternative is to dump the data base at appropriate intervals onto some suitable physical medium which is retained together with the transactions that were processed after the dump was performed. This technique has been employed for the recovery of direct access files in batch processing systems, (Gunton, 1969). In the event of a malfunction the data base is restored from the copy and the transactions processed since the dump are reprocessed. An alternative to reprocessing the transactions is to retain a separate copy of the updated records. The data base is then restored by replacing the relevant records by their updated version. This is similar to the incremental dumping procedure of Fraser (1969) except that in that case the entire file that has been altered is dumped. A slight modification to the foregoing is to not perform insertions or deletions of the records in the

*Now at London School of Economics, Houghton Street, London WC2.

data base between dumps but to have the records of the data base that were added available to transactions as well as the version of the data base that was dumped. The collection of updated records is known as a change file. At suitable intervals the data base and the change file are merged to create a new data base. This data base is then dumped to produce the new backup copy. This approach has the advantage of minimising the exposure of the data base to the duration of the batch merge.

The taking of backup copies for recovery purposes is generally known by the general term 'dumping'. However there are two approaches that may be taken in implementing the copying process. These are:

2.1. *Dump/Restore*

Dump/Restore is the simplest approach in that the data base is copied exactly as it stands onto the medium that will hold the backup copy. The advantages of the approach are its speed, the likelihood of the availability of the program for performing the copying and its easy implementability. The main disadvantage is that because of the indiscriminate nature of the approach errors that exist in the data base are transferred onto the backup copy. An approach that may be taken before dumping is to check all records that have been altered since the last validation. The validation program could pass through the file examining closely all records with a control number greater than that at validation. This examination would involve looking at all the record linkages and checking, where possible, for logical consistency.

Validation after dumping can be performed by employing the dumped copy as the data base that is accessed by subsequent transactions. The original copy of the data base then becomes the backup.

2.2. *Unload/Reload*

Unload/Reload approach strips off a copy of the data base which is in a form ready to be reloaded. This means that the logical structure of the data will have to be taken into account as well as the data. Therefore the time taken to produce a copy will be proportional to the complexity of file organisation. Rather than a simple copy utility program, a retrieval program will have to be employed that retrieves each record. This retrieval program will perform either all or a considerable proportion of the validation of the information held in the master file. All the data and, if the file organisation is complex, most linkages will be checked. Validation is achieved by reloading the master file immediately after performing the unloading. In other words the master file is reorganised at the same time as a backup copy is produced.

3. *Duplication*

Identical versions of the same file are updated in parallel. These files would normally be on separate storage devices.

Only the last two methods are suitable for a transaction oriented (or online) system because the processing mode employed in such a system is random (and in some cases skip-sequential). The generation method of maintaining data base integrity is suitable only for the serial or sequential mode of processing as employed in typical batch processing systems. Successive sections will discuss only dumping or duplication methods of maintaining data base integrity.

Data base recovery with dumping

Since a software or hardware malfunction will occur at some stage in the life of the system it is necessary to define the procedure by which the data base can be restored after such a malfunction. The procedure will be dependent on the method of preserving data base integrity that has been chosen and also on the particular requirements of a transaction oriented environment. Examples are given by Oppenheimer and Clancy (1968), Tonik (1971) and Wilkes (1972).

It has already been stated that to recover from a failure either the messages that caused updates of the data base or the updates themselves must be preserved. They would be preserved on a file called the log file (or journal). These are combined with the dumped copy of the data base to produce the data base as it was at the point of failure. However there is a problem if only the messages are retained. This is because it is not possible to determine what stage had been reached in processing the message when failure occurred. On restart a particular record may be updated for a second time (double updating) or may not be updated at all. If only copies of the updated records are preserved on the log then the integrity of the data base will be maintained as far as the system is concerned but the terminal user will not be certain of the exact state of the file. This is because he cannot be sure whether a message that was successfully transmitted performed an update of the required record. In other words the integrity of the data base is maintained by the system but it may be destroyed by the user. The user may assume that the processing of the message has updated the record when it has not and he therefore does not retransmit the message. Similarly assumption of nonupdating of the record leads to the retransmission of the message which may cause double updating of the record. The problem is exaggerated if a number of records can be updated by a single message and if a multithreading mode of operation is employed.

Consequently to maintain integrity both messages and copies of updated records, the 'after' images, should be preserved. In the event of a failure the data base is restored from the dumped copy and brought up to the state it was in at the time of failure by reading the log file in reverse order to the order in which it was created and replacing the original records by their updated versions. As normal logical addressing of a file extends only to the physical record level, a data change would be recorded by dumping at least the entire physical record which has been subject to update. If the file is very large, one may have to deal with cyclic dumps in which different areas of the data base have been dumped at different times thus requiring additional co-ordination of the after image log. A two-level scheme for dumping and sorting the logs may be employed to minimise reconstruction time (Drake and Smith, 1971). In the first level records are appended to the current log as updates occur and at the same time the logical ID of each updated record is appended to a small auxiliary file. The second level consists of an update of a sorted log file and this job can be run whenever conditions permit. All records updated since the previous update of the sorted log file have their identities recorded (perhaps many times) in an auxiliary file. This file is first sorted by logical address and then the log file update is run with modified records being copied from the data base. To reconstruct the data base the sorted journal can be merged with the last data base dump, and the resulting version updated by the most recent copies of records in those first-level log tapes which have not yet been accounted for by a second-level log update. This sorting reduces the time required for reconstruction in the event of a failure. If the system is operating in a single thread mode of operation then the last message, i.e. the first one read, on the log file is the one that was being processed when the failure occurred. Any copies of updated records caused by the message are not written onto the master file. This is because the exact point during the processing of the message that was reached when failure occurred is not known. On restart the processing of the message is commenced from the start. There is a problem though that it is not possible to determine whether or not the output message for the second last message received has been successfully transmitted Even more important for a system operating in a multi-thread. mode since a number of messages may be being processed when the system fails, is the output message transmission complete indicator to determine from the log which messages were being processed. The order on arrival of messages does not necessarily correspond to the order of completion.

There is a further item of information which if recorded will aid the task of recovery. This is a copy of records that are about to be updated in their preupdate state, the 'before' images. This information will allow recovery from trivial errors due to a program error or the transaction being incompleted without the need for restoring the complete data base from the back-up copy. Trivial errors are defined to be errors that affect only a small proportion of records. The records affected can be restored from the log file alone and reupdated if required also by reprocessing the relevant messages. The 'before' image is the same as the 'after' image of the log tape created after the dump before the last. The main reason for storing both 'before' and 'after' images is to cut down the search time and thus the recovery time. Whether or not this information is recorded will depend on the size of the data base (and hence the time taken to recover) and the time taken to record the information, which will affect the throughput capability of the system.

To summarise, the components needed for data base recovery where dumping is the method of recovery are (*a*) a copy of the data base taken at some convenient point, and (*b*) a log of interactions with the data base since the dump was taken. To allow full recovery so that ambiguities do not exist, the following information should be recorded on the log:

(*a*) input message (or a sub-set that will allow processing to commence)

(*b*) copy of records after they are updated

(*c*) output message transmission complete indicator.

Optionally a copy of updated records before they are updated may be recorded to minimise the time taken to recover from trivial errors.

If the data base itself is not updated but updates are collected on a change file, then the data base is restored solely from its dumped copy while the changes file is restored solely from the log.

Data base recovery with duplication

The procedure for recovery when duplication is employed is simpler than that for dumping.

If there is a software or hardware malfunction and one copy of the data base is corrupted or is inaccessible then recovery consists of simply switching to a mode of operation that involves only the serviceable copy. This copy will be the only version that is subsequently referenced and updated. A problem then arises when the first copy is repaired as it will be out of step with the second. There is a necessity for blocking all accesses to the data base until the repaired copy can be brought up to date by having itself copied to by the serviceable copy.

There would seem to be no need for a log when duplication is employed. However it may be when extreme security is required and a second redundant copy of the data base is maintained by means of the log and a dumped copy. Also a reduced log is required to prevent the sort of ambiguities faced by the user that were discussed earlier and to recover from a failure that occurred during the actual updating operation.

Instead of recording a copy of a record in its after update state the before update copy is written on the log. Therefore the information recorded in the log file is the following:

(*a*) input message

(*b*) copy of the record that is about to be updated

(*c*) output message transmission complete indicator.

When recovering a fault the log file is read backwards as before but only information pertaining to incompleted processing is retrieved. The messages are replaced in the input message queue and the relevant records restored to their original state. On restart those messages that were incompletely processed are reprocessed from the start in their original order of arrival.

Performance

1. *Normal overhead*

In the type of systems being considered (information processing), the factor that will have most effect on the normal performance of the system will be the number of input/output operations. Normal performance is defined to be the performance of the system at times other than at failure or recovery. The overhead due to extra processing will be insignificant due to the relatively low level of processor utilisation. Therefore the difference in normal performance between a high availability system utilising dual processors and a single system will be negligible. Consequently only the performance of the file subsystem will be considered. The system considered will be assumed to employ a log file. There are two components of performance that will be examined: normal performance which is a function of the overhead introduced by the recovery system, and restart time, the time to reconstruct the files after a system failure.

The performance is defined to be the response time of the file subsystem to a particular request. Response time is defined to be the time from the initiation of a data transfer request to the completion of the transfer operation. Mean response time is a function of the following system properties:

(*a*) equipment characteristics

(*b*) equipment configuration

(*c*) file organisation, including block length and the degree to which the most frequently used files are placed on adjacent cylinders

(*d*) system loading, i.e. number of requests per unit time.

The analysis considers the master files to be located on disc storage devices and the log on magnetic tape but extensions to deal with other devices would be quite straightforward.

The type of devices employed in the study is typified by the IBM 3330 for the disc storage device and by the IBM 3420 for the magnetic tape device.

A number of assumptions are made:

1. File requests arrive at each of m disc module queues in identical independent Poisson streams with a mean arrival rate of λ/m.
2. Plot of arm motion distance against time is approximated by a straight line.
3. Queue scheduling is FIFO.
4. Arm motion may be initiated without the availability of the channel.
5. Seek addresses are evenly distributed over the disc cylinders.
6. Direct access file organisation is employed, i.e. one file access per request.

It is well known that the mean response time for a single-server, unlimited queuing system with Poisson arrival distribution is given by the Pollaczek-Khintchine formula.

$$T_q = \frac{T_m}{1-\rho_m}\left[1 - \frac{\rho_m}{2}\left(1 - \frac{\sigma_m^2}{T_m^2}\right)\right] \qquad (1)$$

T_q is the mean file response time, T_m is the mean module service time, ρ_m is the mean module utilisation and σ_m^2 is the variance of module service time.

The module service time, T_m, consists of arm motion time, T_S wait time in the channel queue and service time within the channel T_c. An approximation as shown by Seaman, Lind and Wilson (1966), can be made to module service time by introducing a blocking factor D_i, which represents the channel utilisation due to all other modules except i, the one arriving

$$T_m = T_s + \frac{T_c}{1 - D_i}\left[1 - \frac{D_i}{2}\left(1 - \frac{\sigma_c^2}{T_c^2}\right)\right] \qquad (2)$$

where T_c is the mean channel service time

σ_c^2 is the variance of channel service time

$D_i = \rho_c - \lambda_i T_{c_i}$

ρ_c is the mean channel utilisation and λ_i and T_{c_i} are the traffic rate and the channel service time of the ith module respectively.

The variance of the module service time is approximately

$$\sigma_m^2 = \sigma_s^2 + \sigma_c^2 + (T_m - T_s - T_c)^2 \qquad (4)$$

For a disc storage device, channel service time consists of rotational delay plus information transfer time. The mean rotational delay can be assumed to be the time to complete half a revolution. Information transfer time depends on the physical record size and is rotation time, R, divided by the number of physical records in a track, S.

$$T_{c_1} = R\left(\frac{1}{2} + \frac{1}{S}\right) \qquad (5)$$

For a magnetic tape device the channel service time is block transfer time, B plus interblock gap time, G.

$$T_{c_1} = B + G\ . \qquad (6)$$

Arm motion time can be described by a linear function of the number of cylinders travelled.

$$T_s = a + bn$$

where a and b are constants and n is the number of cylinders travelled. Since it is assumed that requests are uniformly distributed over the discs' cylinder addressed then it can be shown that the mean number of cylinders travelled is approximated by $N/3$ where N is the number of cylinders containing records ($N \gg 1$).

i.e. $$\mathrm{T}_s = a + bN/3 \qquad (7)$$

If λ is the total traffic rate to the disc storage devices in accesses per second then the traffic rate, λ_1, to the magnetic tape device, which contains the log file, will be a function of the proportion of accesses that are writes and the amount of information that is recorded.

i.e. $$\lambda_1 = pl\lambda \qquad (8)$$

where p is the proportion of accesses that are writes and l is the number of records written per write access.

If there are m storage devices ($m - 1$ disc storage devices) then channel utilisation ρc is given by

$$\rho c = \sum_{i=1}^{m} \lambda_i T_{ci}$$

where $$\lambda_i = \frac{\lambda}{m-1}\ \ i = 2, \ldots, m \qquad (10)$$

The overall mean channel service time can be calculated by weighting the two types of service by their frequency of occurrence.

i.e. $$T_c = \sum_{i=1}^{m} \lambda i\, T_{ci} \Big/ \sum_{i=1}^{m} \lambda i \qquad (11)$$

By substituting equations (10), (8), (5) and (6) into (11), the mean channel service time can be obtained.

The variance of mean channel service time can be obtained in a similar way. The variance of channel service time for a disc storage unit is given by $R^2/12$ (uniform distribution). The variance of channel service time for a magnetic tape storage unit is zero.

$$\sigma_c^2 = \sum_{i=1}^{m}\left[\lambda i\, \sigma_{c_i}^2 + (Tc_i - Tc)^2 \sum_{i=1}^{m} \lambda i\right] \qquad (12)$$

The variance of arm motion time is given by $(bN)^2/12$ (uniform distribution).

It is now possible to solve equations (2), (3) and (4) and hence equation (1). This then gives the mean response time for all traffic. However each type of access will differ from the composite response because of different channel service times. This can be adjusted for by subtracting out the mean channel time and adding in the time corresponding to each access type.

i.e. $$Tq_i = Tq - Tc + Tc_i \qquad (13)$$

An example is taken in which the storage devices employed are the IBM 3330 disc and the IBM 3420 magnetic tape. It is assumed for the disc that there are 20 physical records per track. Also $a = 15$, $b = 0.113$ and $R = 17$ milliseconds. (The effect of rotational position sensing is ignored.) For the magnetic tape the records are singly blocked and have a size of 1000 characters. The interblock gap time is 4.8 milliseconds and transfer rate is 100,000 characters per second.

The effect on overall response time of a number of parameters for different configurations are illustrated. The parameters considered are

(*a*) the proportion of file accesses that are writes

(*b*) the amount of information that is logged

(*c*) the rate of file accesses.

Four configurations are examined. They are

(*a*) one disc device and one magnetic tape sharing a channel

(*b*) four disc devices and one magnetic tape sharing a channel

(*c*) one disc device and one magnetic tape each on separate channels

(*d*) four disc devices on one channel and one magnetic tape on another channel.

For any configuration the file access response time for a read,

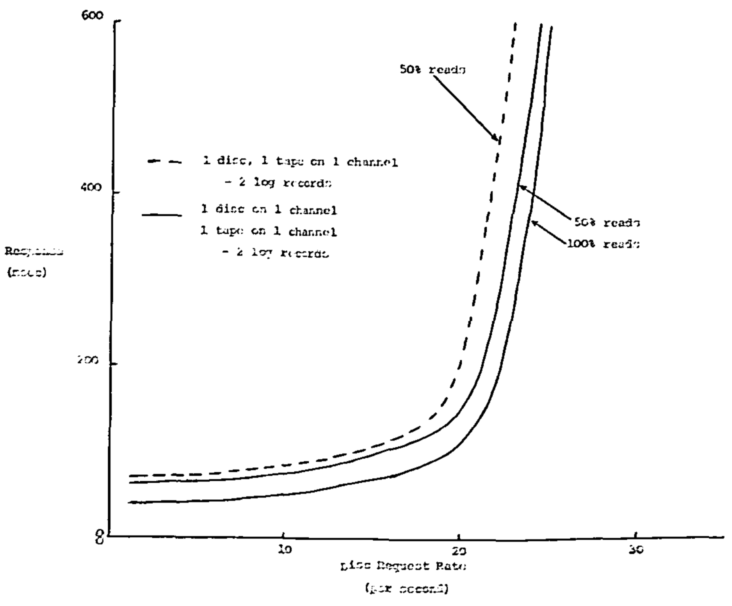

Fig. 1

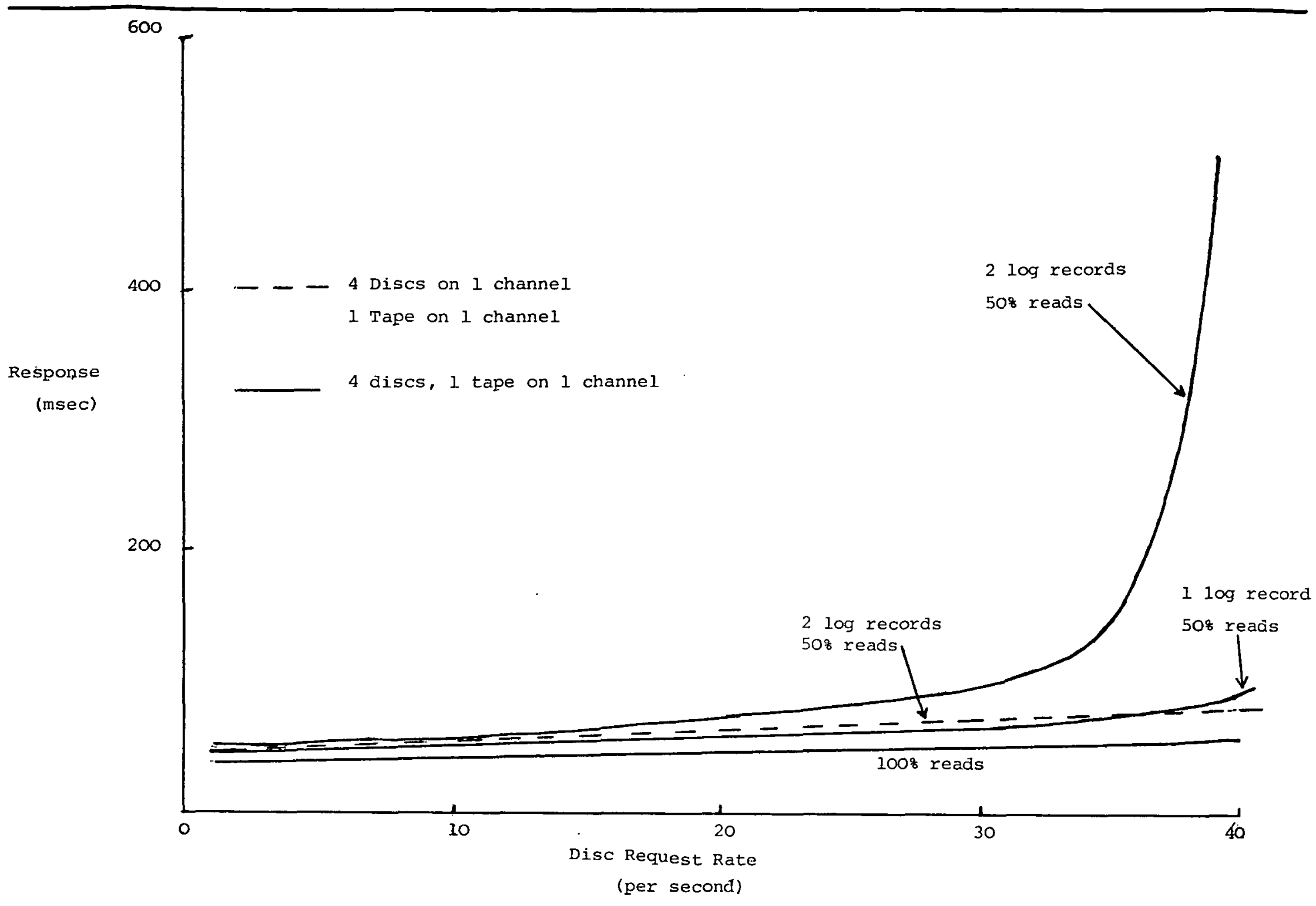

Fig. 2

T_R, is Tq_i and for write, T_w, is $Tq_i + Tq_1$

The overall file access response time is

$$T = (1 - p)\,T_R + {}_pT_w\ . \qquad (14)$$

The different configurations have their main effect on the channel service time for an individual type of request. Where there are two channels there are two separate queues for the file accessing traffic and the logging traffic.

In the example considered the limiting factor on overall performance depends on the particular configuration. For the case of a single disc storage unit (**Fig. 1**), the performance of the disc will be the limiting factor whether or not it has a dedicated channel. If it does have a dedicated channel, the performance threshold is only increased marginally. Also the amount of information logged has little effect on performance limits. For the case of four disc storage units (**Fig. 2**), the provision of a separate channel has a marked effect. If a dedicated channel is provided for the tape unit then the utilisation of the tape unit is the limiting factor. If a dedicated channel is not provided then channel utilisation becomes the limiting factor. As would be expected, the amount of information recorded has more effect on performance for the single disc unit case. For the case of a single channel, having two log records per write request instead of one reduces the throughput capability by nearly 50%.

The performance of the tape unit will have a marked effect on performance in any configuration. For example (**Fig. 3**) if the IBM 3420 is replaced by the IBM 3410 which has a transfer rate of 20,000 characters per second and an interblock gap time of 48 milliseconds, then the overall throughput capability is reduced by over 80% due to the bottleneck caused by the tape unit.

2. *Data base recovery time*

The case of a system employing the dumping method of data base security is considered. Recovery time for a system employing dual copies will be negligible (time for switchover plus reprocessing of current transactions).

The time necessary for recovering when duplicate versions of the files are maintained online is negligible, i.e. of the order of seconds or less.

The size of the log which has to be scanned in the reconstruction of a file is directly proportional to the rate of update and the elapsed time since the last file dump. This assumes that the log file contains only after image copies. The file has to be reconstructed m times between file dumps. In many files as additions and deletions are made, the search cost deteriorates and a reorganisation to reduce the search cost is called for. It would seem economical to perform such a reorganisation when the file backup is created, i.e. it is assumed that the interval between failures is less than the interval between dumps (or reorganisations).

The number of records, n, is assumed to remain constant, i.e. the rate of additions to the file equals the rate of deletion. Also most records can be retrieved in one access but some are overflow records which require two accesses. It is assumed that no records require more than two accesses. The number of overflow records is n_0. Shneiderman (1973) has dealt with the optimum reorganisation interval problem.

The average search cost for retrieving a record whose entry address is known is

$$\begin{aligned} C &= ((n - n_0)/n)b + (n_0/n)2b \qquad (15)\\ &= b(1 + (n_0/n)) \end{aligned}$$

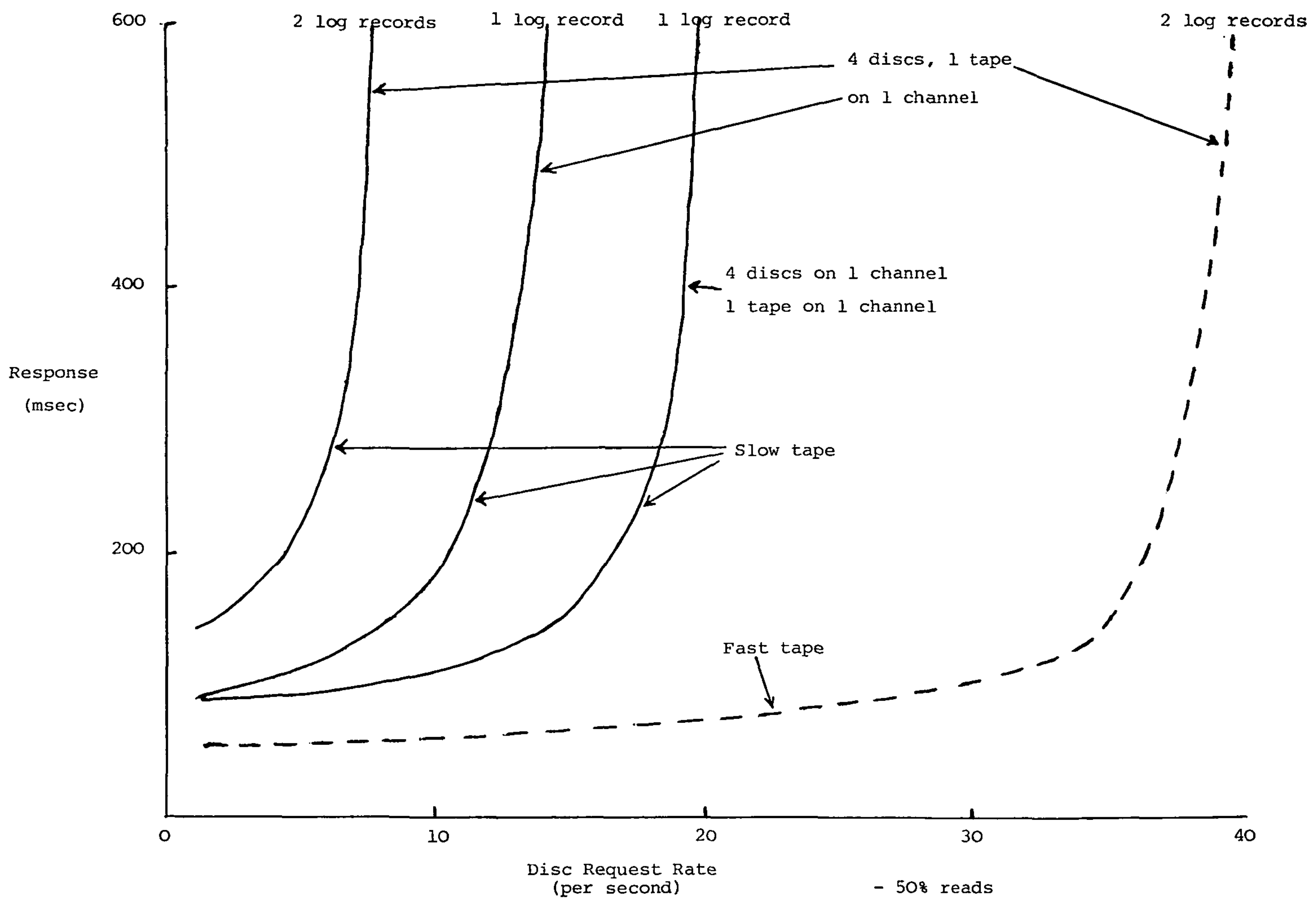

Fig. 3

The loss due to not reorganising is

$$L(t) = \int_0^t (vt'b/n)n_s\,dt' = (vbn_s/n)(t^2/2) \qquad (16)$$

where v is the rate of increase of overflows. The cost of reorganisation, R is a function of the time since the last reorganisation and can be estimated as the retrieval time for all records plus the time to write every record again. Thus cost of reorganisation is cost of retrieval plus cost of rewrite

i.e. $$R(t) = (nb + n_0b) + nb = 2nb + vtb\ . \qquad (17)$$

It is assumed that the average time between failures requiring reconstruction is a and that t is some multiple of a. The cost of reconstruction is proportional to the number of updates. If it is assumed that the number of updates is a constant fraction, p, of the number of requests then the rate of update is given by pn_s $(pn_s > v)$.

The cost of reconstruction, E, is a function of the time since the last reorganisation (or dump) and can be expressed as the time to read and write the records held on the dumped copy plus the time to read and write the log. It is assumed that the average search cost for each access is b for both the dump file and the log file. Extensions to deal with different search costs are quite straightforward. Therefore $E(i)$ the cost of the ith reconstruction is given by

$$E(i) = 2nd + 2pn_sabi\ .$$

The total reconstruction cost between reorganization is given by $(t = ma)$.

$$E(t) = \sum_{i=1}^{m} [2nb + 2pn_sabi$$
$$= (2nb)(t/a) + pn_sb(t + a)(t/a)] \qquad (19)$$

Summing the total loss from inefficient searching, the cost of reorganisation and the cost of reconstruction for one interval, the cost is given by

$$C(t) = (vbn_s/n)(t^2/2) + (2nb + vtb)$$
$$+ (2nb)(t/a) + pn_sb(t + a)(t/a) \qquad (20)$$

If the system is to run for a length of time $T = Nt$, then the total cost is

$$C(t) = \sum_{i=1}^{N} b(vn_s/2n + pn_s/a)t^2 + 2n$$
$$+ (v + pn_s + 2n/a)t$$
$$= Tb(vn_s/2n + pn_s/a)t + 2n/t \qquad (21)$$
$$+ (v + pn_s + 2n/a)$$

Taking the derivative with respect to time setting the result to zero

$$dC/dt = vn_s/2n + pn_s/a - 2n/t^2 = 0\ .$$

Solving for t

$$t = 2n/(vn_s + 2pn_sn/a)^{\frac{1}{2}}\ . \qquad (22)$$

If there were no insertions or deletions v would be zero and we have the result obtained before by Drake and Smith (1971)

$$t = (2na/pn_s)^{\frac{1}{2}}\ . \qquad (23)$$

Acknowledgements
The author wishes to thank his colleagues at SCICON and the LSE Systems Research Group for their advice and assistance. It is wished to make particular mention of Mr. F. F. Land and Mr. S. J. Waters of the Statistics, Mathematics, Computing and Operational Research Department, London School of Economics.

References
BROWNE, P. S. (1972). Computer Security—A Survey, *Data Base*, Vol. 4, No. 3, pp. 1-12.
CANNING, R. G. (Editor) (1971). Security of the Computer Centre, *EDP Analyser*, Vol. 9, No. 12, pp. 1-15.
CANNING, R. G. (Editor) (1972). Computer security: Backup and recovery methods, *EDP Analyser*, Vol. 10, No. 1, pp. 1-15.
COLLMEYER, A. J. (1971). Database Management in a multi-access environment, *The Computer Journal*, Vol. 4, No. 6, pp. 36-46.
DRAKE, R. W., and SMITH, J. L. (1971). Some techniques for file recovery, *Australian Computer Journal*, Vol. 3, No. 4, pp. 162-170.
FLORENTIN, J. J. (1974). Consistency auditing of databases, *The Computer Journal*, Vol. 17, No. 1, pp. 52-58.
FRASER, A. G. (1969). Integrity of a mass storage filing system, *The Computer Journal*, Vol. 12, No. 1, pp. 1-5.
GUNTON, A. (1970). Recovery procedures for direct access commercial systems, *The Computer Journal*, Vol. 13, No. 2, pp. 123-126.
HOFFMAN, L. J. (1969). Computers and privacy: A survey, *Computing Surveys*, Vol. 1, No. 2, pp. 85-103.
MARTIN, J. (1967). *Design of Real Time Computer Systems*, Prentice-Hall.
MARTIN, J. (1974). *Security Accuracy and Privacy in Computer Systems*, Prentice-Hall.
OPPENHEIMER, G., and CLANCY, K. P. (1968). Considerations for software protection and recovery from hardware failures in a multi-access multi-programming, single processor system, *Fall Joint Computer Conference*, Vol. 33, pp. 29-37.
SEAMAN, P. H., LIND, R. A., and WILSON, T. L. (1966). On teleprocessing system design: An analysis of auxiliary-storage activity, *IBM Systems Journal*, Vol. 5, No. 3, pp. 158-170.
SHEMER, J. E., and COLLMEYER, A. J. (1972). Database sharing: A study of interference, roadblock and deadlock, *ACM SIGIDET Workshop on Data Description Access and Control*, pp. 147-163.
SHNEIDERMAN, B. (1973). Optimum data base reorganisation points, *CACM*, Vol. 16, No. 6, pp. 362-365.
TONIK, A. B. (1971). Recovery of on-line data bases (Panel), *ACM National Meeting*, pp. 103-112.
WAGHORN, W. J. (1968). Shared files, *File 68*, International Seminar on File Organisation, Copenhagen.
WILKES, M. V. (1972). On preserving the integrity of data bases, *The Computer Journal*, Vol. 15, No. 3, pp. 191-194.

Book review

Combinatorial Programming: Methods and Applications, edited by B. Roy, 1975; 386 pages. (*D. Reidel Publishing Company*, $36·00)

This is the proceedings of a NATO Advanced Study Institute held at Versailles in September 1974. The contents are largely concerned with graph-theoretic problems, branch and bound programming and set partitioning. The authors and titles of the individual papers are:

Part I General methodology
H. Müller-Merbach. Modelling techniques and heuristics for combinatorial problems.
P. Hansen. Les procédures d'exploration et d'optimisation par séparation et évaluation.
P. L. Hammer. Boolean elements in combinatorial optimisation.
G. B. Dantzig and C. B. Eaves. Fourier-Motzkin elimination and its dual with application to integer programming.

Part II Paths and circuits
B. Roy. Chemins et circuits: énumeration et optimisation.
M. Gondran. Algebra and algorithms.
N. Cristofides. Hamiltonian circuits and the travelling salesman problem.
J. Krarup. The peripatetic salesman and some related unsolved problems.
J. F. Maurras. Some results on the convex hull of the Hamiltonian cycles of symmetric complete graphs.
F. Glover and D. Klingman. Finding minimum spanning trees with a fixed number of links at a node.

Part III Set partitioning, covering and packing
E. Balas and M. W. Padberg. Set partitioning.
R. E. Marsten. An algorithm for large set partitioning problems.
J. Fréhel. Le problème de partition sous constrainte.
M. W. Padberg. Characterisations of totally unimodular, balanced and perfect matrices.
J. Edmonds. Some well-solved problems in combinatorial optimisation.

Part IV Other combinatorial programming topics
D. de Werra. How to colour a graph.
I. Tomescu. Problèmes extrémaux concernant le nombre des colorations des sommets d'un graphe fini.
D. de Werra. A few remarks on chromatic scheduling.
A. H. G. Rinnooy Kan, B. J. Lageweg and J. K. Lenstra. Minimising total costs in one-machine scheduling.
E. L. Lawler. The quadratic assignment problem: a brief review.
P. Hansen. Fonctions d'évaluation et pénalités pour les programmes quadratiques en variables 0-1.
C. Sandi. Solution of the machine loading problem with binary variables.
H. Müller-Merbach. The role of puzzles in teaching combinatorial programming.

The papers form a very mixed bag: some of them are good or at least respectable, while others would probably have been rejected by referees if they had been submitted to journals in the usual way. It is a pity that this habit of publishing conference proceedings in book form has grown up. It bypasses the normal filtering system by referees, it deflects good papers from the journals which are regularly scanned, it imposes an additional financial burden on libraries, and it often (though not in this case) results in delays in publication. (Amongst the papers in this collection, the Dantzig and Eaves paper had previously been published in the *Journal of Combinatorial Theory*).

There is more theory than practice in this book. Some of the authors take the trouble to investigate whether their methodology or their algorithms would actually work if put on a computer; but, taken as a whole, the book is not an adequate guide or introduction to sound computing practice in this field.

J. M. HAMMERSLEY (Oxford)