Recovery procedures for direct access commercial systems

A. Gunton

SPL International, 75 Grosvenor Street, London W1

Direct access peripherals used for mass-storage purposes present peculiar data recovery problems. Fraser (1969) has described the solution arrived at by Cambridge University for their multi-access system. This article discusses the different approach required for commercial systems: it is hoped that the paper will stimulate those who are implementing such systems to give specific examples of recovery procedures used in practice, the frequency with which recovery is required and any consequential costs or difficulties from the service angle.

(Received May 1969)

Data loss or corruption in direct access systems is potentially more significant in its effects than in magnetic tape systems, especially when 'update-in-place' techniques are used. Only by such techniques can one particular advantage of a direct access peripheral be exploited: that to update one section of a file it is not necessary to reproduce all of it. Reliance on the simple but clumsy expedient of regularly dumping all data onto backing store, usually magnetic tape, and reloading whenever necessary, implies the belief that hardware, software, system design or operator failures will not occur frequently enough to justify the cost of designing and implementing recovery procedures within systems. The main purpose of such procedures must be both to limit machine and operator time lost recovering from system failures, and to control the replacement of corrupted data. The onus of development at present lies on system designers, since manufacturers are so far unable to supply any general purpose software, presumably because requirements depend so much on system detail.

Basic requirements

It is, however, possible to generalise the problems involved to a certain extent. Let us postulate a single main file—a customer billing file, or a large corporation's stock-file—which is on line during working hours for random interrogation, and which is updated daily by batch processing runs. This file will be large enough to make daily dumping of its contents an unnecessarily tedious business, and, without this, recovery operations in the event of any type of system breakdown during the updating runs will seriously derange schedules that attempt to make full and continuous use of the machine. For the purpose of simplicity it has been further assumed that only one program is allowed to update the main file at any one time.

For reasons which will be explained later, standard check-point and re-run procedures, designed to limit the extent of re-run operations in serial access systems, are usually inadequate in direct access systems, but the principle is still valid. Assuming also that, like check-point and re-run procedures, the recovery system should be as nearly as possible invisible as far as the main system is concerned, typical requirements would be:

1. It should enable the system to absorb failures when time is of greatest importance, i.e. during the working day, so that permanent corrective action, if necessary, can be taken when convenient.
2. It should interfere as little as possible with normal running of the system, also in terms of the peripherals it uses, since these, rather than the processor, are likely to constitute the limiting factor.

and an additional practical consideration:

3. It should make maximum use of existing software, since skilled programmer labour is likely to be in short supply.

Given these requirements, the primary decision that must be taken is how much information loss can be sustained temporarily in the interests of continuous running. It is assumed that general reliance can be placed on the hardware, so that, in a live and tested system, major corruption or loss of data will be rare, and that when this happens corrective action will in any case have to be taken immediately. Means will therefore be required, written into updating systems, of detecting and preventing operator mistakes, and of recording and reporting minor hardware failures, so that decisions can be taken about their significance, i.e. whether recovery should be immediate or can be postponed until convenient. Obscure software and system bugs will presumably reveal themselves in their own particular fashion, and may or may not allow the luxury of a decision of this type, but at least the system should provide the option.

Checkpointing

Where serial processing media are in use immediate recovery from errors of this kind is a straightforward

operation, invariably catered for by manufacturers' software, and requiring recording the contents of store and the positions of all peripherals at selected intervals. Restart is effected by re-loading core store with its recorded contents and resetting peripherals, either by program or by instructions to operators (for printers, card-readers, etc.). Indeed, provided individual runs are not too long, one can settle for holding past generations of input files, and repeating earlier runs whenever errors become apparent. The inherent nature of direct access files, when used in the most efficient manner, makes either of these procedures inadequate, since both require that data on input files shall be the same at the time of breakdown as it was at the point one wishes to return to and recreate, perhaps under slightly different circumstances. If a direct access file is 'updated-in-place', i.e. the same physical area is used for both input and output, the system gains because only those parts of the file that are updated need to be changed, but a loss in security is sustained because a complete copy of the file on a particular day is not available for record purposes as a natural consequence of the updating process. Furthermore, as soon as the first record is updated during a processing run the file has been changed and is no longer suitable for re-input to the same processing run unless steps are taken to record either the contents of records before updating or, alternatively, the fact that particular records have been updated.

A 'trace' of updating of this type could be used in conjunction with standard checkpointing procedures, either to inhibit re-processing of these records or to re-instate them as they were at the point at which the run is being restarted, in the same way, effectively, as the contents of store are re-instated by the restart program. The former method is economical in its use of peripherals and processor, but in practice the restraints it places on the system are such as to allow its use only in the rarest of cases. After the restart utility has reset core and peripherals, the updating program is entered normally, but processing is only simulated until the point is reached at which breakdown occurred. The first requirement is that this point should be exactly identifiable from the previously stored 'trace' of processing; the second that the simulated processing should be able to reconstruct its previous operations, from the *updated* main file, sufficiently accurately to move output tape files forward to their former positions, and to reconstruct blocks in store if output is blocked. In practice, this must rarely be possible.

The second method has the merit of simplicity and can conveniently be packaged with the manufacturer checkpoint and restart routines, allowing the main system to run its own separate course. The routines normally allow for entry to a 'preparation' routine after restart, and within this the 'pre-update' copy file would be used to reverse the effects of processing undertaken between the restart point and the breakdown—updated records would be over-written, deleted records re-inserted and vice versa. The amount of peripheral storage space allocated for this 'pre-update' copy depends on the degree of security required. If it is not considered necessary to be able to restart from the beginning of the run, it can depend on the checkpointing frequency, or, conversely, the checkpointing frequency can depend on the space available being filled, or half-filled, thus allowing a choice of two previous restart points. Limiting the scope of the restart facility in this way means that it will not cover certain situations, such as, for example, an output tape being damaged near load point, but if hardware considerations dictate it, the risk must be taken and minimised in other ways. Further sophistications to reduce the volume of data stored can be introduced if convenient.

Pre-update trace

The sequence in which main file and pre-update trace are up-dated is critical, and is detailed in block diagram form in **Fig. 1.** The trace must at all times be complete and either concurrent with or in advance of main file processing. This means that:

1. Records cannot be blocked in store.
2. Successful transfer to the storage medium must be confirmed before the main file is updated.
3. The last record must always be identifiable.

There are two obvious ways of fulfilling this third requirement, each with its own advantages and disadvantages. An independent count or key field can be maintained, and used on restarting to locate directly or recognise the last record of the trace. Alternatively a marker can be held in the last record itself: thus every

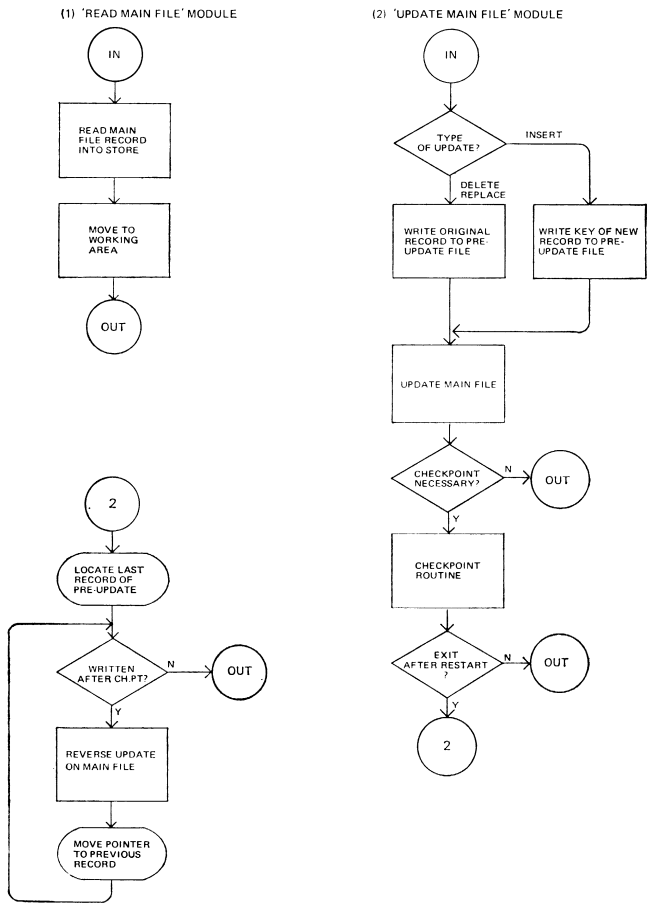

Fig. 1. Block diagram of checkpoint/restart

time a record is written to the 'pre-update' file, the previous record must be re-written without the 'last record' indicator. The former method is clearly more economical in terms of data transferred, but is likely to involve head movement time on a disc, unless it can be held on a separate volume from both main and pre-update files. A further notable advantage is that it can enable direct location of the last record of a file. When complex patterns of deletion and insertion of records are likely, confusion may well arise unless the main file is 'de-updated' in reverse of the original processing sequence.

Fig. 1 also shows how the complete system can be incorporated within 'read main file' and 'update main file' program modules.

Data loss or corruption

Turning to the problem of hardware failure, which may range from head crashes on discs or card wrecks on card files to parity errors in a single block, the optimum situation is where software allows the program under execution to determine the extent of the damage and act accordingly. If it seems appropriate to continue, movements which cannot be processed can be written to a separate file which is re-input the subsequent run and is thus processed normally as soon as errors have been cleared. It can serve also as a record of information loss to be used to determine which sections of the main file to re-constitute from stored information. This method of re-cycling is also a convenient means of ensuring that intermittent failures do not cause movements to be processed out of sequence, if this is likely to be of significance. Provided that movements are dated and held in date sequence they will be processed in the correct sequence as soon as failures are cleared.

The simplest means of recovering data lost through hardware failure, or indeed data corrupted by system errors beyond hope of re-generation is the large-scale dump and recovery operation mentioned earlier. A file that was updated daily could be dumped weekly, and updating runs undertaken since the dump repeated, meaning a maximum of five updating runs after reloading. If one is fortunate enough to have a system in which updating of one record on the main file does not depend on the contents of other records being current with it, the re-loading and re-updating could be restricted to corrupted sections of the file alone, but such systems are rare. Both methods have the advantage that little extra programming effort will be required, possibly only facilities in the updating programs to suppress output if not required during re-runs, but time involved could be considerable, since loss of any part of the file will normally necessitate regenerating all of it.

This factor points the advisability of keeping a record of the results of processing input movements, rather than the movements themselves. Here 'results' does not necessarily imply complete updated records, since it will only rarely happen that the entire contents of a record changes as a result of movements. In this respect main file design can be of significant importance: variable composition records containing coded details of their contents lend themselves to a system whereby only sub-records that are changed and contents indicators need be stored. It may also be possible to combine the functions of pre- and post-update copies of records in a single file, so constituted that, presented with a version of the record before updating, it introduces updating information, and vice versa. With this method it will be necessary to write a separate updating program which selects information stored in one of these forms, and reconstitutes records that have been replaced, this latter

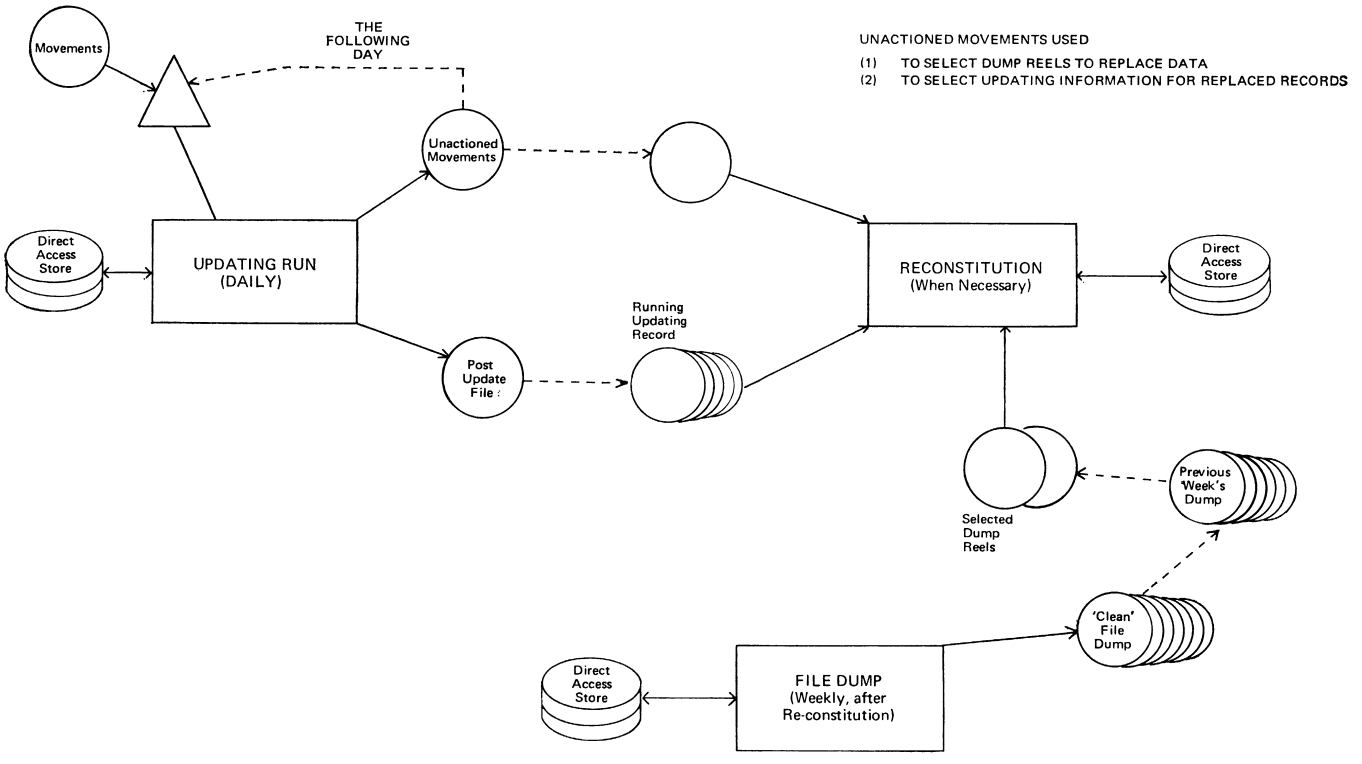

Fig. 2. Example reconstitution system

normally being possible by means of standard software. Control of the operation can be achieved by using a record of information loss output from updating runs to determine selection of records.

The total system is summarised in **Fig. 2,** which shows particularly the dual function of this 'unactioned movements' file. Data cycled in this way will, of course, not necessarily all be movements unactioned because of hardware failure, but those that are can be designated in a particular way and picked out by the reconstitution program. When records are blocked on the main file, one record being unreadable must also imply that those blocked with it are inaccessible, and account must be taken of the fact. Depending on the data organisation, it may in fact be easier to reconstitute a larger area still, possibly a cylinder, because data originally in a home area has been transferred to an overflow area or vice versa. Since all the up-date history has to be read in any event, this will make little difference to the time required for the operation.

General points

If the main file falls easily into logical sections that can be dumped separately on a cyclical basis, the work load can be spread more evenly, dumping, say, a quarter of the file every week. But it brings its attendant problems if used in conjunction with the above method. Not only will a complete updating record have to be maintained as far back as the oldest dump quarter that is current, but also updating information selected during reconstitution only if it is successive to the dump from which a particular record was replaced. Possibly the updating record could be purged of all information that has become redundant, whenever a section of the main file is dumped, but this seems unnecessarily involved for the small advantage it procures. It is preferable in terms of simplicity and machine usage to dump the complete file and start the updating record from scratch every time.

No mention has been made in the preceding descriptions of reporting procedures, which must be regarded as essential. Full details of data loss not only aid decisions about when to reconstitute, but also enable action to be taken outside the main system if delay is critical. With indexed files, it should be possible to ascertain the precise extent of data loss due to hardware failure, but if the manufacturer's software regards indexes as a private affair between itself and the hardware, a copy in more accessible form can always be maintained elsewhere. This is, in any case, advisable, since loss of part or all of the index can render large sections or the whole of a file immediately inaccessible.

Beyond these large-scale security arrangements, more controls will be needed within programs that update direct access files, to guard against operation errors. The consequence of operators starting a run with old input data, for example, can be to necessitate reconstitution of large sections of a file if not prevented by programmed controls. For instance, a dummy account could be held on the main file, and updated at the beginning of each run, containing details of input files previously processed and general information about the state of the system. By reference to this, attempts to input apparently out-of-date information could be queried and prevented before permanent damage was inflicted on the main file. The possibility of this happening, or of hardware or software proving fallible can be ignored if it does not seem to justify the extra systems and programming effort required, but it should be recognised as a dangerous gamble under the present state of development of manufacturers' operating systems.

Reference

FRASER, A. G. (1969). Integrity of a mass storage filing system, *The Computer Journal*, Vol. 12, No. 1, pp. 1–5.

Book review

Topics in Interval Analysis, by E. R. Hansen (editor), 1969; 130 pages. (*Oxford University Press*, £2.50)

This book is an account of lectures given by invited speakers at a symposium on interval analysis sponsored by the Oxford University Computing Laboratory in early 1968. The book is divided into two distinct parts.

Part 1 consists of a description of interval analysis used to obtain error bounds for computed solutions to standard algebraic problems such as the solution of linear and non-linear equations and the inversion of matrices. A description is also given of Triplex-Algol, a formalized language specially devised to cope with interval analysis algorithms. The contributors are R. E. Moore, K. Nickel, E. Hansen and J. Meinguet.

Part 2 deals with interval analysis applied to continuous problems. These include numerical integration, the numerical solution of two point boundary value problems, initial value problems for systems of ordinary differential equations, and partial differential equations. There is also a section devoted to statistical distributions of errors applied to linear programming. The authors here consist of R. E. Moore, E. Hansen, F. Krückeberg and M. Dempster.

Considering the complexity of the subject, the book is particularly easy to read. This has been achieved by the individual authors concentrating on simple examples to illustrate the methods for bounding errors in the various computed solutions. The conclusion to be drawn from this book is that interval analysis has met with considerable success in the analysis of errors for the numerical solution of algebraic problems. The same cannot be said, however, with regard to continuous problems. The material is particularly thin with regard to differential equations and it is difficult to see how interval analysis can have much impact on the numerical solution of partial differential equations in the foreseeable future.

The book is highly recommended and the editor has done a good job in producing a review of recent progress in the fascinating subject of interval analysis.

A. R. MITCHELL (Dundee)