A note on data base integrity

By A. BENCZÚR and A. KRÁMLI

The interest of the authors in the problem of data base integrity came from a practical task. In 1974 we began to build up a new management information system for the Danube Iron and Steal Works. The data base of this system is placed at the CDC—3300 computer of the Hungarian Academy of Sciences and the system can be accessed through a user terminal UT—200. The processing of the data base is random batch processing with some query capabilities, and the problem of its integrity is very close to that of the online processing.

Based on this experience we are going to give a model for describing and analysing data bases from the point of view of the integrity.

Different approaches to the data base management problems (e.g. Codd's relational data base [1], the network data system recommended by the CODASYL DBTG [2]) first of all give models for describing the structure of data bases, or give proposals for standardizing languages, structures and procedures used in data base management systems (DBMS). A very sophisticated DBMS (based on CODASYL DBTG) is described in the paper of Barbara M. Fossum [3], where one can find a list of routines maintaining and preserving the integrity of the data base. The most general problems of integrity are also described there, and we shall use her terminology like rollback, rerun, recovery, and so on.

In the sequel we shall give a very simplified — but a completely sufficient to treat the problem of integrity — model for physical realization of data bases. Using this model we can investigate the dynamic behaviour of the data base.

Physically the data base is a sequence of elementary data items, (in this paper the elementary data items are the least adressable units on the auxiliary memory devices, i.e., the blocks or pages on the disk).

The physical state S of a data base is given by the sequence $\{P_1, \ldots, P_n\}$ of page-adresses and by the sequence $\{C_1, \ldots, C_n\}$ of their contents. In our definition the state of a data base, besides the data in traditional sense, includes also the programs handling them.

Definition 1. We shall say that a state $S'=\{\{P'_1, \ldots, P'_m\}, \{C'_1, \ldots, C'_m\}\}$ is a substate of $S=\{\{P_1, \ldots, P_n\}, \{C_1, \ldots, C_n\}\}$ $(S' \subseteq S)$ iff there exists a subsequence $\{i_1, \ldots, i_m\}$ of indices such that $P_{i_j}=P'_j$ and $C_{i_j}=C'_j$ for every $j=1, \ldots, m$.

Definition 2. A state $\bar{S}$ is an extension of S iff $S \subseteqq \bar{S}$.

The physical state of a data base has to satisfy some rules, which define the syntax, accessing and processing capabilities and so on. We need some further definitions. Let F be a class of "state valued" functions $f(S)$ defined on a set of physical states. The above mentioned rules can be expressed in terms of class F.

Definition 3. We shall say that a state S is consistent with respect to the class F iff $\forall f \in F \ f(S) = S$.

(E.g. if f is a function corresponding to a sort procedure, then S is consistent with respect to f iff it is sorted).

Definition 4. A state S' is called preconsistent with respect to class F, iff there exists a function $g \in F$ such that

(i) for each $f \in F, \quad f(g(S')) = g(S')$,

(ii) for every g' satisfying (i),

$$g'(S') = g(S).$$

The state $g(S')$ is called the *consistent reorganization* of the preconsistent state S'.

Definition 5. A substate S'' of a preconsistent state S' is a generator, iff it is also a preconsistent state, and they have the same consistent reorganization.

Notice that the inclusion $S'' \subseteqq S$ is not necessary.

The above definitions are suitable not only to describe the static state of a data base, but they give a tool to characterize the dynamic one too. Designing a data base we have to construct the class of functions F in such a way that it would assure the following two capabilities.

Assumption 6. (i) The class F must contain functions which carry out the various update procedures. In terms of our definitions, we consider the state of the data base, before initiating an update run, to be consistent.

(ii) We need to assure with great probability the recovery of the data base in the case of loss or failure of some substate of it. For this reason we have to build up some optimal system of nontrivial generator substates.

Up to this point we have assumed that the preconsistent state becomes consistent in a unique step. But physically it cannot be realized. The real update procedures work in the core memory and the state of the data base changes page by page. So, the realization of a function $g \in F$ is a sequence $S_0, S_1, \ldots, S_n = g(S_0)$ of states. (The content of core memory is not included in the notion of a state.)

The defect during an update run, which occurs with a considerable probability, leads usually to the loss of information in the core memory. This raises a special safety problem. In our language, to defend the base against this type of information loss, we have to guarantee the states $S_1, \ldots, S_{n-1}$ to be preconsistent. This condition on the state provides the possibility of rerun and in many practical realizations this possibility is combined with rollback facilities on the quick recovery level.

For the evaluation of cost function of different update procedures we make the following assumption: there is a subset of indices $i_1, \ldots, i_l$ and a sequence of generators

$$S'_{i_1} \subseteqq S_{i_1}, \ldots, \quad S'_{i_l} \subseteqq S_{i_l}.$$

The substates $S'_{i_1}, \ldots, S'_{i_l}$ are the rerun generators and the corresponding functions g_{i_j} determine the rerun procedure.

$$(S_{i_j} = g_{i_j}(S'_{i_j})).$$

We assume, that the processing cost is a monotonically increasing nonlinear function of the run time, so in order to calculate the avarage cost we have to determine the probability distribution function of the run-time.

This can be carried out under some conditions on the failure process and the process of changing-points of generators by standard methods of reliability theory.

As an illustration we show how to calculate the probability distribution function of the extra run time caused by random failures occured during the process $S_1, \ldots, S_n$. We suppose that the following assumptions are fulfilled.

Assumption 7. (i) the failure process $\{\tau_k\}$, where τ_k is the time interval (the length of the time between two consecutive changes is taken for the time unit) between the $(k-1)$-th and k-th failure, is a sequence of independent identically distributed (i. i. d.) discrete valued random variables with common distribution function F,

(ii) the first two moments of distribution F are finite,

(iii) the differences $i_k - i_{k-1}$ are equal to a constant d for every $k=1, \ldots, l$,

(iv) the rerun procedure $S'_{i_j} \to g(S'_{i_j}) = S_{i_j}$ does not need extra time.

Let us introduce the random processes $\{\theta_k\}$ and $\{\varepsilon_k\}$ as follows:

$$\theta_1 = \max_{i_l < \tau_1} i_l,$$

$$\vdots$$

$$\theta_k = \Big(\max_{i_l < \sum_{j=1}^{k-1} \theta_j + \tau_k} i_l \Big) - \sum_{j=1}^{k-1} \theta_j,$$

$$\varepsilon_k = \sum_{j=1}^{k-1} (\theta_j + \tau_j) - \sum_{j=1}^{k} \theta_j = \tau_k - \theta_k.$$

The random variable θ_k denotes the time interval between two consecutive regeneration points of the update run, and ε_k the extra run time caused by the k-th failure.

The following two properties are consequences of conditions

(i)—(iv) assumption and the definitions of processes $\{\theta_k\}$, $\{\varepsilon_k\}$:

(v) $\{\theta_k\}$ forms a sequence of i. i. d. random variables having finite first and second moments,

(vi) $\{\varepsilon_k\}$ forms a sequence of i. i. d. random variables bounded by d.

We have to determine the probability distribution function of the random variable $\eta_n = \sum_{k=1}^{\nu_n - 1} \varepsilon_k$, where ν_n is the first moment m for which $\sum_{k=1}^{m} \theta_k \geqq n$. The asymptotic behaviour of the distribution function of random variables η_n, when $n \to \infty$, can be expressed by the following theorem:

Theorem 1. If $F(d) \neq 1$, then $E(\theta_1) \neq 0$ and

$$\lim_{n\to\infty} P\left\{ \frac{\eta_n - \frac{n}{E(\theta_1)} E(\varepsilon_1)}{\sqrt{\frac{n}{E(\theta_1)} E(\varepsilon_1 - E(\varepsilon_1))^2}} < x \right\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{x^2}{2}}.$$

Proof. It follows from (v) and the weak law of large numbers that for $n \to \infty$, $\frac{\nu_n}{n} \to \frac{1}{E(\theta_1)}$ in probability. Thus taking into account property (vi) we can apply to $\{\eta_n\}$ Anscomb's central limit theorem for the sum of a random number of random variables (see Rényi [4]).

The difference between the long and quick recovery problems can be summarized as follows: in the quick recovery we preserve generator substates of preconsistent states during an update run, while for solving the long recovery problem we have to preserve (generally duplicate) generator substates of a consistent state before a sequence of update runs. Between two duplications we collect the changes in such a way, that the preserved generators of the original state and the changes together compose a generator of the present state.

So the reliability of the system is the product of the reliability of the preserved generator and that of the preserved changes. The cost of this recovery system consists of the cost of the periodical duplications, the cost of collecting the changes (proportional to the update time) and the average cost of the recovery. The probabilistic treatment of this process is analogous to that of the quick recovery.

Finally, we mention some special problems arising in the use of the operating system MASTER of computer CDC—3300. (See [5]). We must handle the SCHRATCH-pool as an extension of core memory and not as a part of the physical state. The only exception is the INP-file for not DIRECT input JOB-s. In this case the operating system ensures the restart of the JOB using the INP-file as a part of a generator, when an AUTOLOAD has occured during the run of the JOB.

This facility was exploited only for automatic restart of update processes by preserving only the control cards. The input data were put into the data base before initiating the update system in strict sense.

This method can serve to collect the changes for long time recovery and in some cases to obtain a part of a generator for quick recovery.

The update system of the realized DBMS mentioned at the beginning of our paper consists of 40 processes corresponding to the various types of data. Each process is organized in such a way that their repetition, or rollback, or continuation when it is interrupted would be available. The organization of the correct runs of these processes is solved by a special program (TASK), which calls the update tasks in correct sequence, determines the input data, controls their run and, in the case of a restart calls the necessary rollback procedures and continues the run from the interrupted task.

The long recovery is based on the File Back Up system of MASTER.

Abstract

The integrity and reliability problems of data base systems arise in computer praxis and theoretical investigations too. The authors give a statistical model for the description of data bases, which is sufficient to treat the integrity problem.

COMPUTER AND AUTOMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
H-1502 BUDAPEST, HUNGARY

References

[1] CODD, E. F., A relational model of data for large shared data bases, *Comm. ACM*, v. 13, No. 6, 1970.

[2] *CODASYL committee data base task group report*, Association for Computing Machinery, April 1971.

[3] FOSSUM, B. M., Data base integrity as provided for by a particular data base management system, *Proceedings of IFIP Working Conference on Data Base Management*, Cargese, Corsica France, 1—5 April 1974.

[4] RÉNYI, A., On the central limit theorem for the sum of a random number of independent random variables, *Acta Math. Acad. Sci. Hungar.*, v. 11, 1960, pp. 97—102.

[5] CDC 3170—3300—3500 computer systems MASTER version, 4.0. *Reference Manual CDC*, 60415100, No. 1973.

(Received March 11, 1976)