PUC

Series: Monografias em Ciência da Computação
Nº 3/79

SELECTIVE RELOADING OF VERY LARGE DATABASES

by

Daniel A. Menascé

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225 - CEP 22453
Rio de Janeiro — Brasil

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação
Nº 3/79
Editor: Daniel A. Menascé March, 1979

SELECTIVE RELOADING OF VERY LARGE DATABASES*

Daniel A. Menascé

* This research was partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, under contract 1885/78.

Resumo:

A propriedade de um sistema de gerência de banco de dados de ser capaz de recuperar-se de falhas é extremamente importante. Alguns tipos de falhas, tais como erros permanentes de memória secundária, requerem que o banco de dados seja recarregado a partir de uma versão previamente armazenada ou "dump". Esta operação pode consumir muito tempo em se tratando de um banco de dados muito grande. Este trabalho apresenta uma técnica através da qual não é necessário recarregar todo o banco de dados mas somente algumas porções do mesmo. Estas porções devem ser, cuidadosamente, selecionadas a fim de que a integridade semântica do banco de dados seja preservada. Esta técnica se baseia num modelo formal de recuperação de erros desenvolvido por Menascé et al [MENA 79]. O artigo apresenta uma possível implementação para o modelo de recuperação.

Palavras chave:

banco de dados, recuperação de erros, "dumps", recarga seletiva, integridade semântica, falhas, erros.

Abstract:

The property of a database management system to recover from failures and maintain the database integrity is extremely important. Some types of failures, like media failures, require that the database be reloaded from a previously saved copy or dump. This operation may be extremely time consuming for very large databases. This paper presents a technique by which it is not necessary to reload the whole database but only portions of it. These portions must be carefully selected so as not to destroy the semantic integrity of the database. This technique is based on a general formal model of crash recovery developed by Menasce et al [MENA 79]. An implementation for the recovery model is also given here.

Keywords and phases:

database, crash recovery, dumps, selective reloading, semantic integrity, failures, errors.

SELECTIVE RELOADING OF VERY LARGE DATABASES

Daniel A. Menascé
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brasil

1. Introduction

The property of a database management system to recover from failures and maintain the database integrity is extremely important. Failures may be of different kinds and they require different corrective actions. In this paper we assume that users interact with the database via transactions which are the units of consistency and recovery [GRAY 78 and LAMP 76]. It is important to distinguish failures from errors [GRAY 78]. An error is the part of the system which is incorrect while a failure is the agent which causes the error. Errors in a database may be classified as permanent or temporary. A temporary error is one which is detected as soon as it is caused and can be fixed by undoing some actions of some transactions. A permanent error is one which requires that the database be reconstructed from a previously saved copy or dump. Permanent errors may be caused by media failures, DBMS failures and errant transactions.

Taking a dump may be a rather time consuming operation. With current technology, it would take of the order of ten hours to dump a database of ten billion bytes [CIBB 76]. Therefore, in many applications which require high availability of the database it is not possible to take the database offline in order to dump it to a removable storage media. Some techniques for dynamically dumping a database have been proposed by Rosenkrantz in [ROSE 78]. Another technique which can be employed to circumvent this problem is the use of differential files [SEVE 76].

When a permanent error is detected the database must be reloaded from its dumps. This operation may also be extremely time consuming. In this paper we will present a technique by which it is not necessary to reload the whole database but only portions of it. These portions must be carefully selected so as not to destroy the semantic integrity of the database. This technique builds on the formal model of crash recovery developed by Menascé et. al. [MENA 79].

This paper is organized as follows. Section 2 presents the elements of the recovery model. The properties of the model are explored in the following section. Section 4 presents a possible implementation of the recovery model, including a description of the system modules and of the data structures which are necessary for its operation. The next section introduces the procedure that must be followed when errors are detected. The paper concludes with the presentation of an op-

timization problem relevant to the model.

2. Recovery Model

We describe here the basic elements of the crash recovery model considered in this paper.

A transaction is a sequence of actions delimited by a BEGIN and END commands. The actions of a transaction may be READ, UPDATE (modify, insert or delete), LOCK or UNLOCK commands. For our purposes we are only interested in the UPDATE action since it is the only one which modifies the contents of a database. It is also assumed that each transaction has a unique transaction identifier, denoted TR_ID. A transaction is the unit of consistency [ESWA 76] in the sense that it takes the database from a consistent state into another consistent state. A transaction is also the unit of recovery [LAMP 76] in the sense that it should be regarded as an atomic unit, i.e. either all or none of its actions are reflected into the database.

Another element of our model is a logical subdatabase (LDB). We assume that the database is partitioned into logical subdatabases LDB1, LDB2, ..., LDBk such that the union of all the LDB's is equal to the whole database. A logical subdatabase must be described in a value independent manner since we do not want the LDB's to vary with the contents of the database. In a relational database this implies that a logical database could be described in terms of whole relations or domains of relations but not as a subset of tuples of a relation. Figure 1 gives an example of a relational database, DB, and a possible partition into seven logical subdatabases.

```
DB  : EMP(emp#, empname, job, salary)
      PROJECT(pname, manager, budget,
                        supporting_agency)
      TASK(tname, pname, tsupervisor)
      ASSIGNMENT(tname,emp#)

LDB1 : EMP(emp#,job)
LDB2 : EMP(emp#,empname,salary)
LDB3 : PROJECT(pname,manager)
LDB4 : PROJECT(pname,budget,supporting_agency)
LDB5 : TASK(tname,pname)
LDB6 : TASK(tname,tsupervisor)
LDB7 : ASSIGNMENT(tname,emp#)
```

Figure 1 - Partition of a DB into Logical Subdatabases.

We will not allow value dependent specification of LDB's such as

LDB : EMP(emp#,salary) where salary > 10K.

Another element of our model is a dump of a logical subdatabase. These dumps are taken independently from one another and they are timestamped. There is also a log of transactions which is used to reconstruct the database from the dumps as will be explained in a later section.

We are now in a position to introduce the reload graph, Gr. This graph has two kinds of edges, directed and undirected. There is a directed edge associated with every dump of a logical subdatabase. These edges are called dump edges. There is an undirected edge associated with each transaction. These edges are called transaction edges. The rules which define the graph are the following:

1. there is a directed path in Gr for each LDBi containing all the dump edges associated with LDBi. These dumps appear in increasing order of their timestamps.

2. dump edges are labeled with dump names. A unique dump name may be obtained by concatenating the LDB name with the timestamp of the dump. Each node in the path associated with a logical subdatabase is labeled with the name of the unique dump edge incident out of it. The last node in such a path is called terminal node and is labeled with the name of the corresponding LDB followed by an asterisk.

3. let t be a transaction which updates data items in LDB's LDBi1, LDBi2, ..., LDBij and let di1, di2, ..., dij be the dumps of LDBi1, LDBi2, ..., LDBij which immediately precede the execution of transaction t. Then, all the nodes into which di1, di2, ..., dij are incident into are linked by a path of transaction edges.

4. there is a node called, source node, which is connected via directed edges to the first node of each directed path.

5. by assumption, there is an initial dump of each LDB. These dumps are taken before any transaction is run.

Figure 3 illustrates the graph Gr for the LDBs in figure 1 and for the types of transactions listed in figure 2.

3. Properties of the Reload Graph

T1 : give a salary increase to all managers
(involves LDB1 and LDB2)

T2 : fire all the managers of the projects which are supported by supporting agency XYZ and which have less than 5 tasks allocated to them.
(involves LDB3, LDB4 and LDB5)

T3 : give a salary increase to all supervisors of task T.
(involves LDB2 and LDB6)

T4 : assign employee # 105 to the task which has Smith as supervisor.
(involves LDB6 and LDB7)

Figure 2 - Example of some transactions.

Some definitions will be given here along with the most important properties of the reload graph Gr. The properties are stated without proof since they follow from the fact that we are using the model introduced by Menascé et. al. in [MENA 79].

Definition 1 : (Reload Set):

A set, S, of dumps of logical subdatabases is called a reload set if it is the minimal set such that the database is brought to a consistent state when the dumps in S are used to reload the corresponding portions of the database.

Definition 2 : (Optimum Reload Set):

Let S = {LDB1, ...,LDBn} be a set of LDB's detected to be in error. The optimum reload set of S is the reload set which contains the most recent possible dump of each LDB in S.

This is the set that we are interested in. The graph Gr has the following properties.

P1: A cutset of Gr formed of dump edges only is a reload set.

P2: All the dump edges in a cycle of the underlying undirected version of Gr are useless since they cannot appear in any reload set. Therefore, they should be discarded.

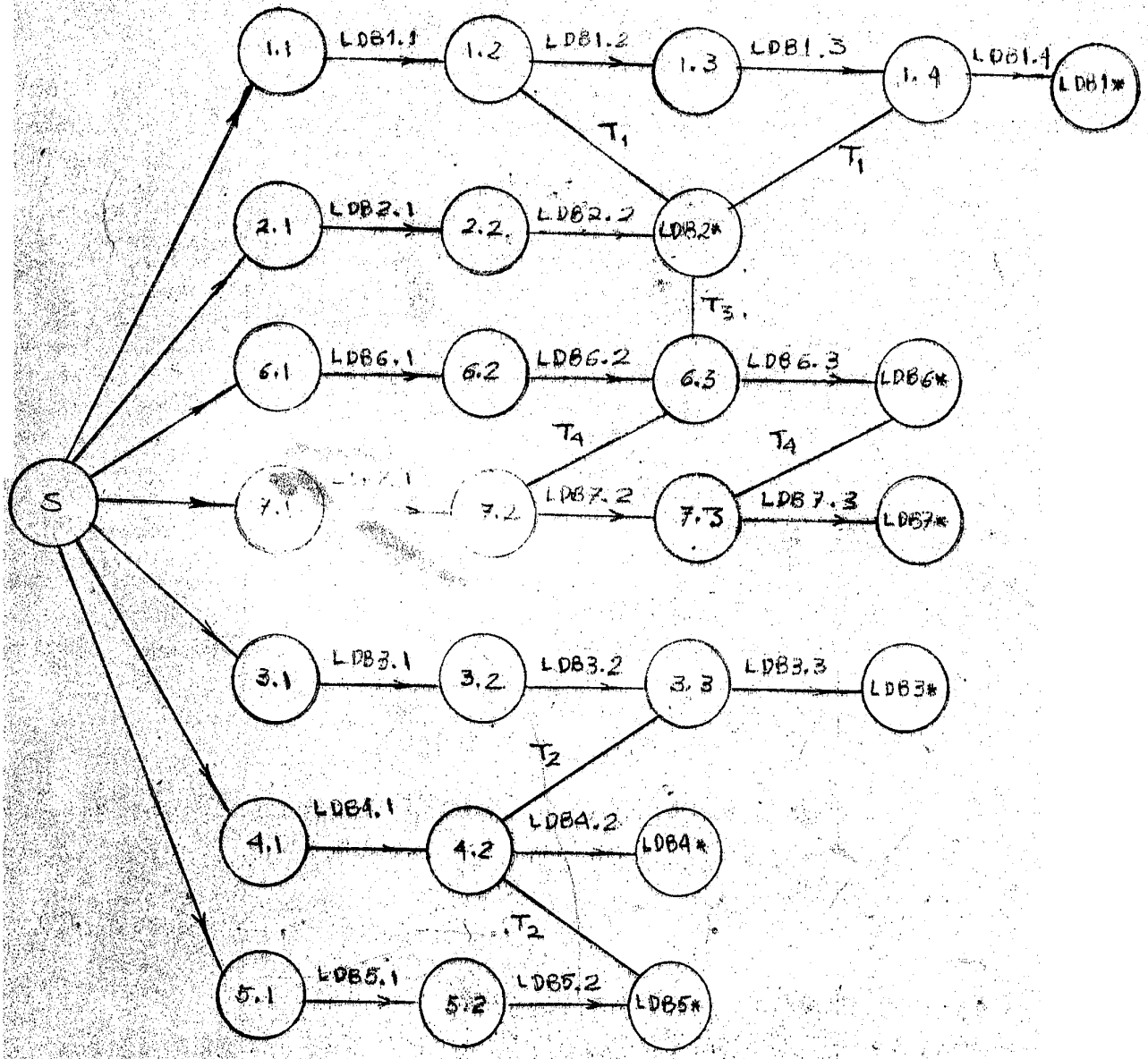


Figure 3 - Reload Graph Gr.

Since reload sets contain no transaction edges, the graph can be condensed by combining into a single node all the nodes which are linked through a path of transaction edges. The resulting graph will be called condensed reload graph. Hereafter we will only be using the condensed graph and we will refer to it as reload graph for short.

Figure 4 represents the condensed version of the graph

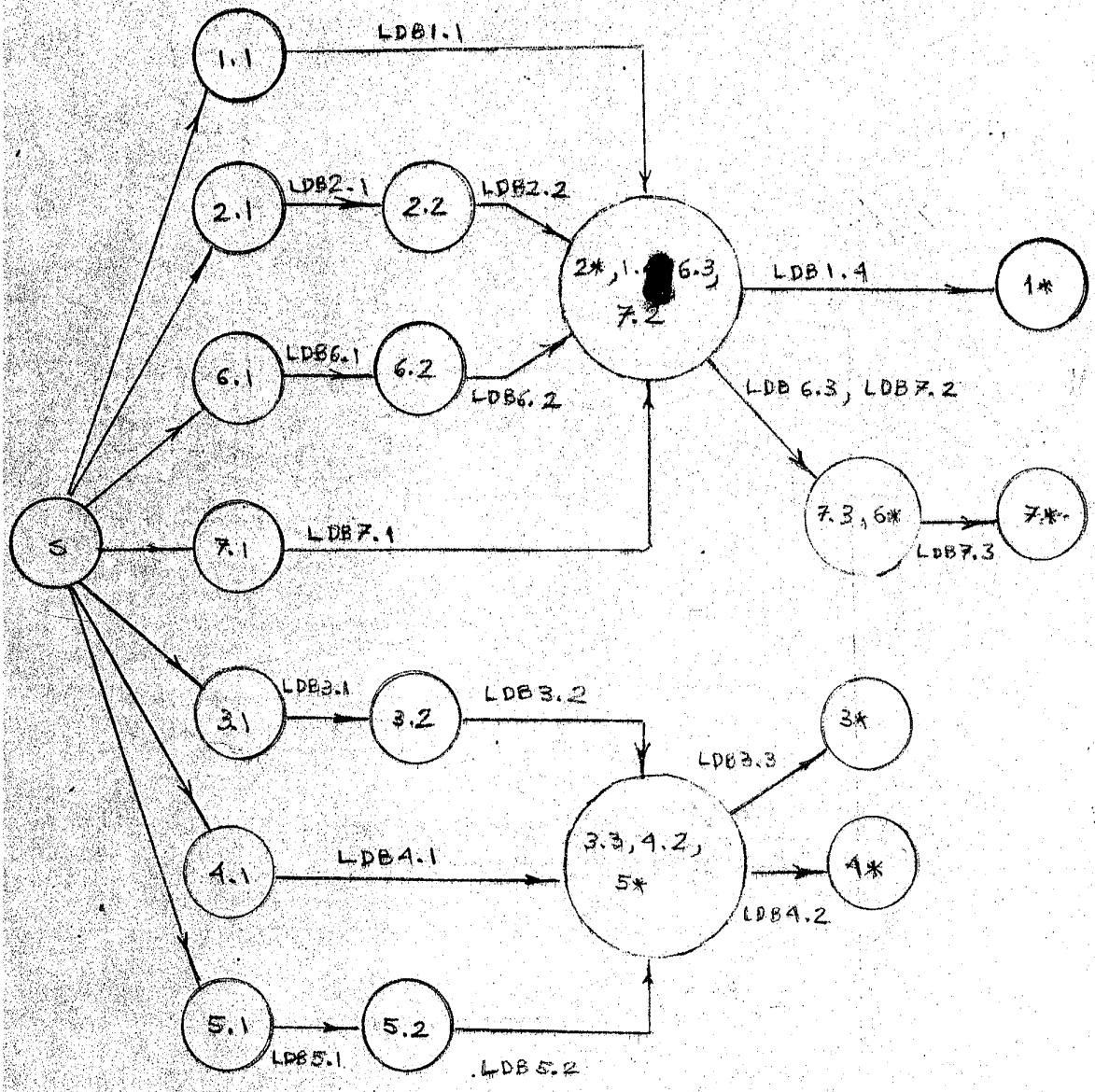


Figure 4 - Condensed Reload Graph

in figure 3.

Properties P1 and P2 can now be rephrased as follows:

P1': A cutset of the condensed reload graph is a reload set.

P2": All the edges in a cycle of the underlying undirected version of the condensed reload graph Cr represent useless dumps which should be discarded.

In a later section it will be shown how to calculate the optimum reload set of a reload graph.

4. Implementation Considerations

This section presents a possible implementation for the model suggested in the previous sections of this paper. The system is composed of four modules, namely:

1. COMMAND ANALYZER
2. BEGIN PROCESSOR
3. UPDATE PROCESSOR
4. END PROCESSOR

These modules are functionally related as indicated in figure 5.

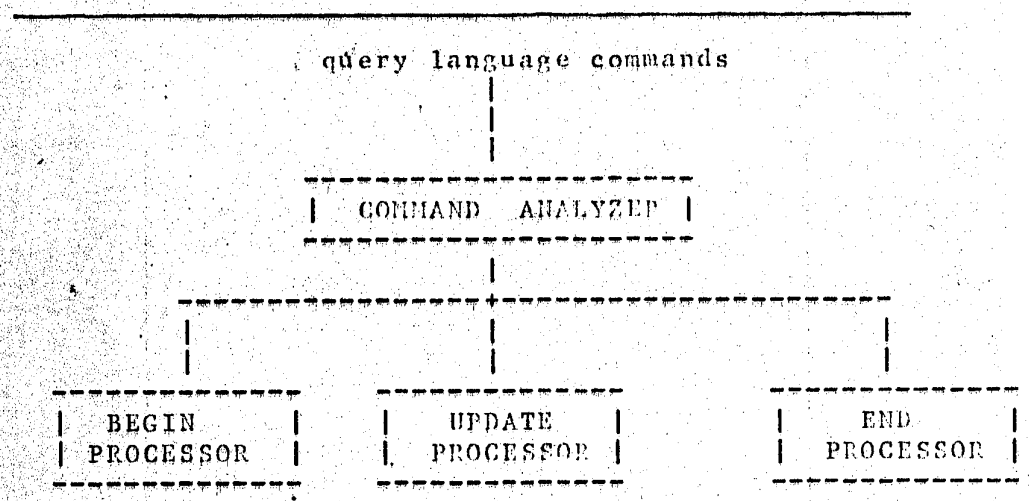


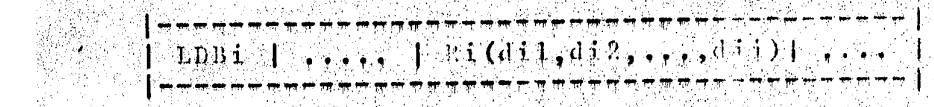
Figure 5 - System Modules

The input to the COMMAND ANALYZER is the set of query language commands submitted by the user to the DBMS. These commands are separated into BEGIN, UPDATE (modify, insert or delete) and END commands which are passed to the BEGIN PROCESSOR, UPDATE PROCESSOR and END processor respectively. Next section describes the necessary data structures.

4.1 Data Structures

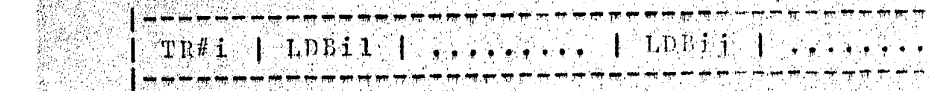
. Logical Subdatabase Description Table (LDBDT)

This table describes each LDB in terms of its component relations and domains. There is an entry in the LDBDT for each logical subdatabase LDBi which is of the form



. Active Transaction Table (ATT)

This table has an entry for each active transaction. The form of each entry is shown below



and it indicates that the transaction with TR ID equal to TR#i is active and has so far modified data items in the logical subdatabases indicated in the table entry.

. Transaction Log

The transaction log is a sequential file which provides a continuous historical record of the processing activity of a system [CIBB 76]. The transaction log contains all the information usually found in similar logs existing in most DBMSs in addition to the list of LDBs updated by the transaction.

4.2 Command Processors

. BEGIN Processor

The BEGIN Processor receives a unique transaction identifier, TRi, from the COMMAND ANALYZER and creates an entry in the ATT for transaction TRi and sets the list of modified LDB's in this entry as empty.

. UPDATE Processor

The UPDATE Processor receives a complete update command from the COMMAND ANALYZER. It then parses it in order to extract the names of relations and domains which are going to be modified by the command. The LDBDT is then searched to determine which are the LDB's which have as components the relations and domains extracted from the update command. The

list of these LDB's is appended to the corresponding entry in the ATT.

. END Processor

When the END command of transaction TRi is received by the END processor, the following actions are taken.

1. Collapse the terminal nodes (if more than one) of the condensed reload graph, Gr, corresponding to the LDB's indicated in the ATT entry of transaction TRi.

2. Look for cycles in Gr. If any is found, discard all the dumps associated with edges in the cycle (cycles) and collapse all the nodes of a cycle into a single node.

3. Check policy regarding minimum number of dumps to be maintained for each LDB. If, as a reult of action 2 above, this policy is violated for any LDB, new dumps should be generated to compensate for it.

4. Write a log record corresponding to transaction TRi. This record should contain the list of updated LDB's in the ATT entry of transaction TRi.

5. Delete from the ATT the entry corresponding to transaction TRi.

5. Recovery Procedure

The recovery procedure consists in finding the optimum reload set and to reload the dumps of the logical subdatabases specified in it. In order to describe how the optimum reload set is calculated some definitions are necessary. Let x be the terminal node(LDBi). Let Gr be the reload graph. Let $G^*(LDBi) = (V,E)$ be the graph induced in Gr by the set of nodes V given by

$$V = \{x\} \cup \{v \mid \text{there is a path from } x \text{ into } v \text{ in } Gr\}$$

Let Y = LDBil,...,LDBij be a set of LDB's. Let T(Y) be the set of terminal nodes of the LDB's in Y. Let $G^*(Y) = (V(Y),E)$ be the graph induced in Gr by the set of nodes V(Y) defined as

$$V(Y) = T(Y) \cup \{v \mid \text{there is path from } y \text{ in } T(Y) \text{ into } v \text{ in } Gr\}$$

Given these definitions, the optimum reload set of a set Y = {LDBi1,...,LDBij}, denoted ORS(Y), is given as follows

$$ORS(Y) = \{ v \rightarrow w \mid v \text{ is in } G^*(Y) \text{ and } v \rightarrow w \text{ is not in } G^*(Y) \}$$

In other words, the optimum reload set is the set of dumps associated with edges of Gr which are incident into nodes of G*(Y) but which are not themselves in G*(Y). Therefore, in order to find the optimum reload set one must find first the graph G*(Y). This operation can be efficiently done (in linear time and linear space) by traversing the graph G*(Y) using a depth-first search procedure with starting points in each of the terminal nodes of the LDB's.

Figure 6 illustrates the graph G*({LDB2}). In this example, ORS({LDB2}) = {d11,d22,d62,d71}. Therefore, if an error is detected in LDB2, one must only reload selected dumps of LDB1, LDB2, LDB6 and LDB7.

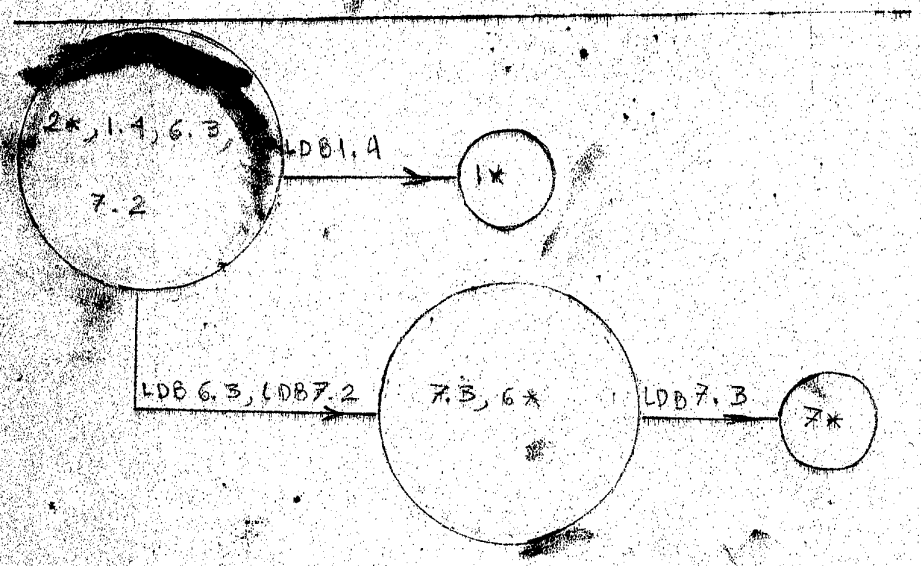


Figure 6 - Graph G*(LDB2)

Once determined the optimum reload set, ORS, one must reload the portions of the database associated with the dumps indicated by the set ORS. The transaction log must now be positioned to a point in time equal to the minimum timestamp of the dumps in the optimum reload set. The log is then read forward and for each log record found the following action is taken.

If the transaction updated at least one of the LDB's specified in the ORS, then the transaction must be reprocessed.

6. Optimization Considerations

The number of LDB's which have to be reloaded depends on how the LDB's are selected. If a proper selection is made, most of the transactions will confine their updates into a single LDB. If the set of transactions which are going to be run against the database is known, an optimal selection can be made. This problem can be stated as follows. Given the data items (relations and domains) updated by each transaction type, given the frequency of occurrence of each type of transaction find a partition which minimizes the frequency of inter-LDB transactions.

7. Conclusion

Reloading a whole database from previously saved copies may be an extremely time consuming operation. A technique which allows for selected portions of the database to be reloaded while preserving the database integrity was presented in this paper. This technique is based on a general formal model of crash recovery [see MENA 79] and a possible implementation for the model was proposed here..

References

GIBB 76 Gibbons, T., "Integrity and Recovery in Computer Systems', NCC Publications, Manchester, UK, 1976.

GRAY 78 Gray, J.N., "Notes on Database Operating Systems', Operating Systems: An Advanced Course, Springer-Verlag Berlin Heilderberg, 1978, pp. 394-481.

LAMP 76 Lampson, B. and H. Sturgis, "Crash Recovery in Distributed Data Storage Systems", Xerox Palo Alto Research Center Technical Report, 1976. (also to appear in the CACM)

MENA 79 Menascé, D.A., R.R. Muntz and G.J. Popek, "A Formal Model of Crash Recovery in Computer Systems", Proceedings of the Twelfth Hawaii International Conference on System Science, Hawaii, January 4-5, 1979.

ROSE 78 Rosenkrantz, D.J., "Dynamic Database Dumping", Proceedings of the 1978 ACM/SIGMOD International Conference on the Management of Data, Austin, Texas, May 31- June 2, 1978, pp. 3-8.

SEVE 76 Severance, D.G. and G.M. Lohman, "Differential Files: Their Application and Maintenance of Large Databases", ACM Transactions on Database Systems, September 1976, Vol. 1, No. 3, pp. 256-267.