State Dependency Issues in Evaluating Distributed Database Availability *

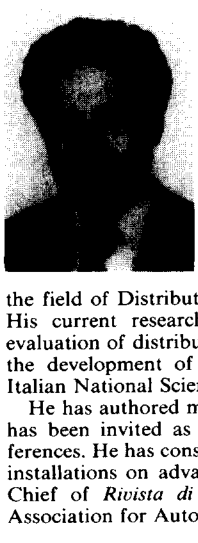
Fabio A. SCHREIBER
Istituto di Matematica, University of Parma, 43100 PARMA, Italy

Quantitative evaluation of availability in distributed database systems must take into account both hardware and software failures. Therefore parameters such as the failure and the recovery probabilities must be evaluated for each "component" of the system.

The most difficult problems arise when the evaluation procedure has to deal with failure mechanisms which functionally depend on the state of the system.

In this paper, after a short introduction to a general methodology for availability evaluation, an example of state-dependent fault mechanism is described together with a technique for evaluating its impact on the life of the whole system.

Keywords: availability, distributed databases, Markovian models, state-dependent failure rates, system dependent failure mechanisms, time-outs.



Fabio A. Schreiber received the Dr. Ing. degree in Electronic Engineering from the Politecnico di Milano in 1969. Since then, he has been with the Computer Science Laboratory at the Dipartimento di Elettronica of the Politecnico di Milano, as an assistant and then as an Associate Professor of Applied Electronics. Since 1981 he is Full Professor of Computer Science at the Mathematical Dept. of the University of Parma.

His main research interests are in the field of Distributed Informatics and Information Systems. His current researches include performance and reliability evaluation of distributed computing systems. He is involved in the development of a distributed DBMS sponsored by the Italian National Science Council.

He has authored more than forty papers on these topics and has been invited as a speaker to many workshops and conferences. He has consulted for several companies and university installations on advanced system design. He is the Editor-in-Chief of *Rivista di Informatica,* the journal of the Italian Association for Automatic Computing.

* This work has been partially supported by the DATANET project of the Italian CNR Progetto Finalizzato Informatica, and by Ministero Pubblica Istruzione (fondi 40%).

1. Introduction

The development of several prototype Distributed Database Management Systems (DDBMS) put into evidence a set of problems bound to the reliability of the distributed system and to the availability of the physical Databases at the different sites.

The goal of obtaining a very high availability of the overall system (i.e. a very high probability that the system operates within a time interval – called the *mission time*) stimulated researchers to design rather sophisticated architectures and related recovery procedures which could survive a number of multiple failures [1,4–6,9].

However, mechanisms to assure availability – besides being effective in their primary goal – should not cause a large overhead, in order to keep a good level of performance. Therefore a good compromise must be attained between the availability for the Distributed Database (DDB) (which for many DDB applications is rather low if compared with the 1 hour stop in 40 years required for telephone switching equipment) and acceptable throughput and response times. To obtain such a compromise, we must be able to quantitatively evaluate the DDB system availability, a research field which has been little explored.

In published research, it is generally assumed that failures are independent from each other. This assumption, even if reasonable as a first approximation in many systems, hides some state dependent failures. These failures, in some cases, can lead to a "domino effect" with catastrophic results for the whole system (e.g. remember the great black-out of the New York area in 1977).

However, while it is possible to conceive some general models which allow for a quantitative evaluation of the availability of a system affected only by independent components failures, things become much more complicated when failures depend also on the system architecture resulting by assembling such components and on the system functional requirements (systemic errors/failures).

We feel that in such cases only some general guidelines can be given to proceed toward the model definition, but the detailed model must be carefully determined case by case. A model for a state dependence case arising in a distributed database with partitioned and duplicated files is described in [12].

This paper shortly reviews in a systematic way a method for the quantitative evaluation of availability in a DDB, which was firstly presented in [11]. Then the method is extended to the case of a state dependent failure mechanism which results from the interaction between functions at very different levels in a DDB Management System. Section 2. reports the main concepts and results presented in [11]. Section 3. deals with the problems resulting from state dependent failures; a case is presented of a systemic failure induced by a time-out mechanism, and queuing theoretic results are used to evaluate the state dependent transition probabilities, which are the base for availability evaluation.

2. Quantitative Evaluation of Availability

Let us define:

availability of the DDB with respect to a transaction Ti
$\mathscr{A}_{Ti}(t)$ = probability that the system performs Ti successfully at time t;

availability of the DDB
$\mathscr{A}(t)$ = probability that the system performs all transactions successfully at time t

Let us suppose now that transactions can be executed independently on disjoint sets of resources. Therefore:

$$\mathscr{A}(t) = \mathscr{A}_{T1}(t) * \mathscr{A}_{T2}(t) * \ldots * \mathscr{A}_{Tn}(t) = \Pi \mathscr{A}_{Ti}(t) \tag{1}$$

Then, our goal is to evaluate the availability of the system with respect to the generic transaction Ti. To this end, the first step is to model the Ti processing path as a set of connected components, which, in our case are constituted by the data items affected by the transaction and by the communication paths linking the remote sites the data are stored at [11]. If the hypothesis of independence for transactions does not hold, the decomposition of the problem expressed by (1) is not possible and the availability of the DDB must be evaluated for the transaction set as a whole, as we shall see in section 3.2.

2.1. System Description

We suppose that a component is either up (state 1) or down (state 0); no other state is allowed. This hypothesis is supported by the nature of many fault detection mechanisms which, in most cases (e.g. time-outs), are binary in nature. Furthermore we shall consider a data item to be available if all the hardware/software components, required for accessing it, are available. We are not considering here the access interference problem for shared data items, which deserves attention by its own [14].

A state of the system (as far as transaction Ti is concerned) is determined as the set of the states of its components. Therefore, given a system with N components, we can describe its states by a 2^N *state vector* of state words, N bits each, ordered in a predefined way.

Let us define *critical failure* those failures or sets of failures of the system components which prevent Ti from being successfully committed.

To describe the state of the system we can associate the State Vector with an ordered binary column vector C_{T_i}, called the *structure vector* [2,3]. Elements of C_{T_i} are zero if the corresponding state represents a critical failure, one otherwise.

As an example, let us consider the following system where four data items $\{x_1,\ldots,x_4\}$ are redundantly stored at four different sites $\{N_1,\ldots,N_4\}$:

- site N_1 contains the database $d_1 = \{x_1, x_2, x_3\}$;
- site N_2 contains $D_2 = \{x_2\}$;
- site N_3 contains $D_3 = \{x_2, x_3\}$;
- site N_4 contains $D_4 = \{x_4\}$.

To represent the relations among the components (e.g. sites and communication lines) needed in processing a particular transaction, we can build an oriented acyclic flow diagram. Branches in the graph represent system components. If two components are both required to process Ti, they are linked in a series branch in the flow-graph. If two components are redundant, they are placed in parallel branches. An initial branch represents the site at which the transaction is entered, and a final

Table 1
State and Structure Vectors

State No	State Vector							Structure Vector
	Host Sites State				TLC Links State			
	$N1$	$N2$	$N3$	$N4$	2–1	2–3	2–4	
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	1	1	0
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
60	0	1	1	1	0	1	1	1
61	0	1	1	1	1	0	0	0
62	0	1	1	1	1	0	1	0
63	0	1	1	1	1	1	0	0
64	0	1	1	1	1	1	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
125	1	1	1	1	1	0	0	0
126	1	1	1	1	1	0	1	1
127	1	1	1	1	1	1	0	0
128	1	1	1	1	1	1	1	1

vertex represents the ending of the transaction either by commitment or by abortion.

For the transaction to be executed there must exist one complete path from the initial branch to the final vertex (i.e., the flow-graph must be connected).

With the flow-graph a boolean expression can be associated in which the states of the components are the logical variables and they take value "0" if the component is faulty, value "1" if it is working. If two components are serially linked in the flow-graph they are connected by an .AND. operator in the boolean expression, while they are connected by an .OR. operator if they are linked in parallel. Then the boolean expression can be minimized with the fundamental theorems of boolean algebra.

Therefore, given a transaction, the evaluation of the boolean expression tells us if a given system state represents a critical failure for it. Then we can say that the State Vector and the Structure Vector represent the truth table of the boolean expression.

A detailed description of how the State and the Structure Vectors are built can be found in [11]. Here we only show in Table 1 a partial instance of the two vectors for transaction T2 started at site 2, reading data items x_3 and x_4, and updating at least two copies of data item x_2 in the example distributed Database. The complete example is worked out in [11].

2.2. System Evolution

Reliability features of a component are often expressed in terms of its MTTF (Mean Time To Failure) and MTTR (Mean Time To Repair) or in terms of their reciprocals: the Failure Rate (FR) λ and the Repair Rate (RR) μ (i.e. the transition rates).

From the definition given at the beginning of section 2, we can compute availability at time t in the following way

$$\mathscr{A}_{T_i}(t) = \boldsymbol{C}'_{T_i} * \boldsymbol{p}(t), \tag{2}$$

where $\boldsymbol{C}'$ is the transposed vector of $\boldsymbol{C}$ and $\boldsymbol{p}(t)$ is a vector whose jth component, evaluated at $t = t^*$, gives the probability that the system be in the j-th state at $t = t^*$.

The evolution in time of $\boldsymbol{p}(t)$ is given by a (discrete) Markov Law

$$\boldsymbol{p}(t+1) = A * \boldsymbol{p}(t), \tag{3}$$

where A is a square matrix of order 2^N called the *transition matrix.* Its a_{ij} element gives the probability the system be in state i at time $t+1$, being in state J at time t. Syntheticully A can be expressed as follows:

$$A = \sum_{k=1}^{2N} \left(\underset{i=1}{\overset{N-1}{\Pi_\otimes}} \boldsymbol{x}^{(i)} \right) * \boldsymbol{e}_k^{2^N}, \tag{14}$$

where

$\Pi_\otimes$ is the Kronecker (or direct) product of two matrices [8];

$\boldsymbol{e}_k^{2^N}$ is the kth versor of a 2^N space, i.e. it is a vector with 2^N components, whose kth entry is a 1 while the others are zeroes;

$\boldsymbol{x}^{(i)}$ is $\left|{}^{f^{(i)}}_{1-f^{(i)}}\right|$ if the ith element is up, while it is $\left|{}^{1-r^{(i)}}_{r^{(i)}}\right|$ if it is down;

$f^{(i)}$ is the probability $\lambda^{(i)} * \Delta t$ of failure during the observation period Δt;

$r^{(i)}$ is the probability $\mu^{(i)} * \Delta t$ of recovery during the observation period Δt.

Equations (1)–(4) allow us to evaluate the availability of the system. For more details and examples see [2,11].

3. The State Dependent Case

Up to now no mention has been made of how transition probabilities are evaluated. If we suppose failures to happen independently from each other, then the transition probabilities appearing in the transition matrix are those of each component taken by itself. However, if failures depend in some way on each other, in the most general case each $f^{(i)}$ and each $r^{(i)}$ must be a function of the overall system state S

$$f^{(i)} = f^{(i)}(S); \quad r^{(i)} = r^{(i)}(S),$$

and the complexity of the problem grows exponentially with the number of components. However, if we can indentify M smaller subsystems in such a way as components belonging to the same subsystem have dependent reliability behavior, while components belonging to different subsystems are independent (and luckily this is rather common), we can apply a "divide and conquer" technique by evaluating M simpler Markov chains p_{S_l}, while the global probability is obtained as

$$\boldsymbol{p}(t+1) = \underset{l=1}{\overset{M}{\Pi_\otimes}} \boldsymbol{p}_{S_1}(t+1). \tag{5}$$

So far as to the evaluation method. However the real difficulty in complex systems as DDB is to determine the functions $f^{(i)}(S)$ and $r^{(i)}(S)$. The difficulty lies in the need of examining a large part of the system and to determine, for each possible failure, the consequences it has on other functions or components. Therefore it will not be possible to find a general methodology, but each type of failure shall be treated by itself.

What we want to show in this paper is that state dependency is really meaningful in a DDB system and that dependencies may be tricky. To do this we shall make use of a case example.

3.1. An Example

An in-depth study and the corresponding implementation in the field of DDB systems reliability have been described in [5]. This system is structured in several different layers, the lowest of them (called *RELNET*) is totally devoted to create a reliable communication system .. and something more. In fact, besides assuring message delivery and possibly their recovery, RELNET performs site state monitoring and it maintains a System Global Time, which is used also by functions and mechanisms at higher levels (e.g., for concurrency control in accessing data items).

The need of obtaining global synchronization

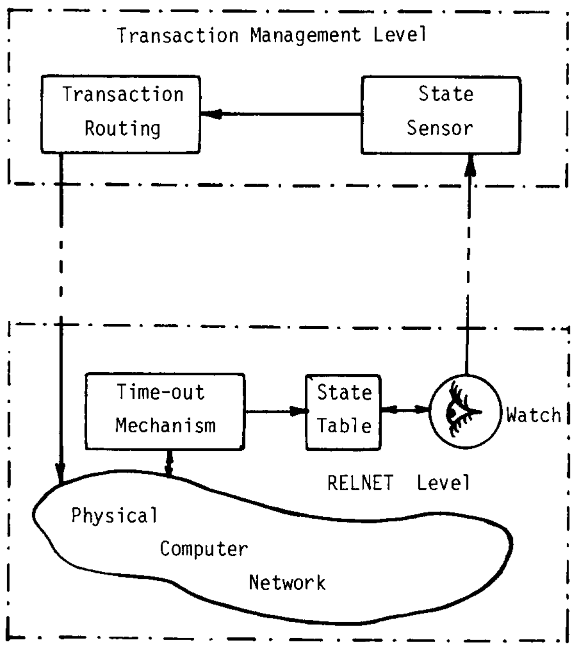


Fig. 1. The positive feedback loop.

poses very tight requirements on the response time of some type of synchronization messages. If the polled site does not answer within a given time, a RELNET mechanism declares it crashed, whatever the reason is for the delay, and a recovery procedure is started. This philosophy is not peculiar of RELNET, but it is rather common in many real-time process control systems, where time plays an essential role in determining the correctness of operation.

One of the causes which can be responsible for an excessive delay is a momentary overloading of the polled site, and it is well known, from queuing theory, that response time increases exponentially with the resource utilization factor.

Let us suppose that at a higher level of the DDBMS a transaction management function take advantage of the existence in the DDB of multiple copies of data items to enhance the system's availability.

If the system is working in a heavy load condition, the case will happen that a site times-out and is declared crashed. A primitive in the reliable network layer (a Watch) will inform the Transaction Manager of the failure; as a consequence, the last will reroute transaction processing from the failed site to other sites where copies of the required data items are available. This will result in an abrupt load increase at sites already working at limit load conditions. Therefore they will slow down their response times, and eventually they will time-out, thus being declared failed. This mechanism is shown in Fig. 1, and while it is inspired by the work on SDD-1, it is not representative of all the sophistications which were included in that system.

It is evident how this positive feedback, unless broken by effective countermeasures, can lead to a Domino Effect [13] and hence to the failure of a large part of the system.

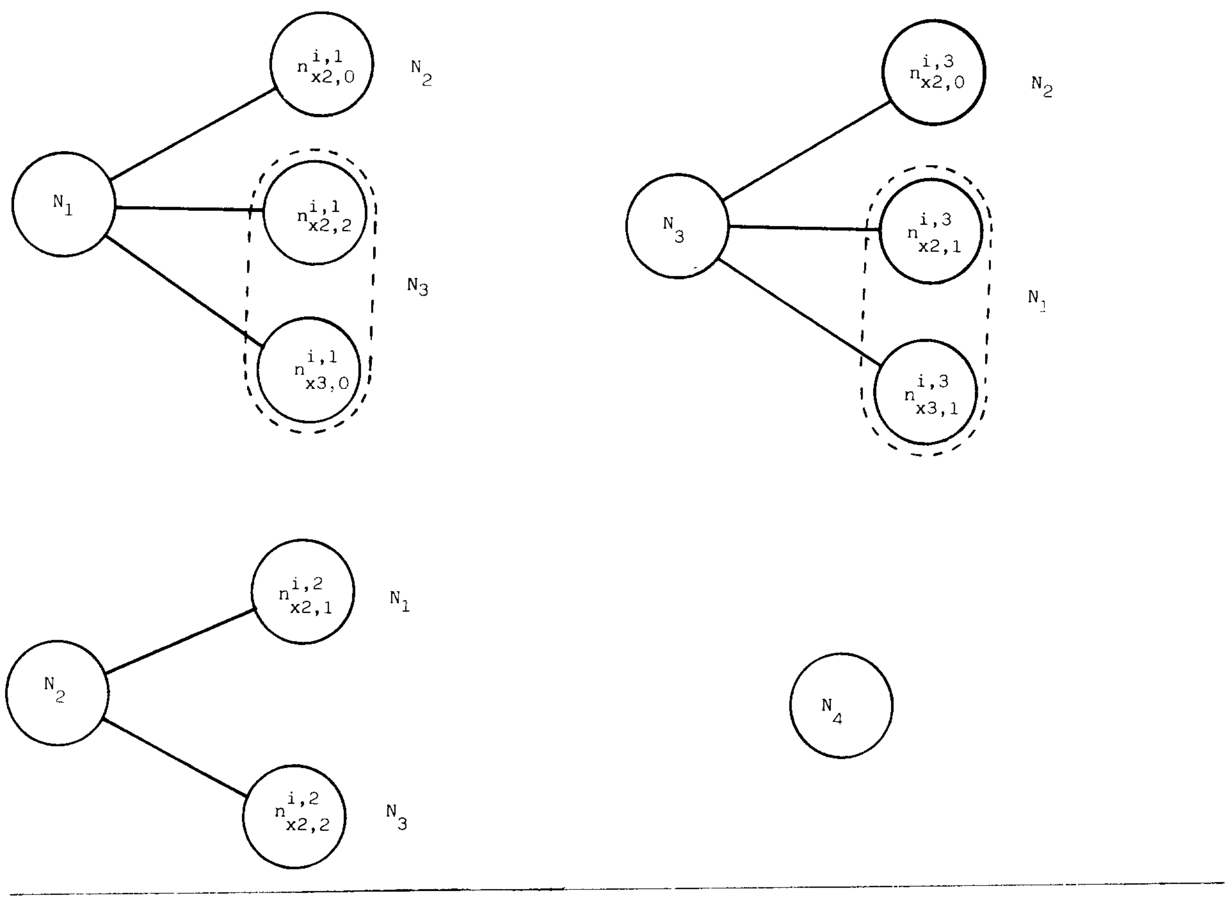


Fig. 2. The structural model for transaction $T2$.

3.2. Evaluation of State-Dependent Transition Probabilities

In this section we are going to determine how the Transition Probabilities of a system component depend on the state of the system as a whole, when the type of failure we want to examine is that presented in section 3.1.

We limit our analysis to site failures as a whole, supposing that communication channel failures are hided by the network's own recovery procedures. However the extension to both site and communication failures, and/or to failures of single parts of the host computers, can be made quite easily, and it affects mainly the state and the structure vectors. Let us state the following:

Hypothesis 1. The failure mechanism described at the end of sec. 3.1 has a propagation delay which is much smaller than the time needed to recover a failed site.

This hypothesis, which is valid with a good approximation for many software systems (tenths of milliseconds against several seconds), allows us to consider the system as being "without repair". Therefore we can limit ourselves, at least in a first time, only to the evaluation of the failure probabilities $f^{(i)}(S)$ and of the probability that, after some time, the system will reach a critical failure state.

Let us suppose that a transaction *Ti*, originated at some site, is to access data items stored at some other sites and that some of these data items are redundantly stored in multiple copies. For the sake of simplicity, we can define a static routing strategy for accessing the data items required by each transaction. If there is just one copy of the data item – needless to say – this will be chosen. For multiple copy data items we can associate each transaction with a priority table indicating, for each data item, which is the first choice copy to access, which the second choice (or first back-up), etc. What is a back-up copy for a transaction can be a primary for another one; so we must suppose there is some activity for each physical data item. How to choose priorities in order to obtain good strategies is not in the scope of this paper.

Firstly, we must build the State and Structure Vectors, introduced in section 2.1. For this purpose, let us partition the sites of the system in the following way:

$N = \{N_1, \ldots, N_N\}$, the set of system sites (host computers);

$N^{(i)} = \{N_1^{(i)}, \ldots, N_M^{(i)}\} \subseteq N$, the set of sites relevant to transaction *Ti*;

$N^{i,f} \subset^{(i)}$, the set of "back-up" sites for site $N_f^{(i)} \in N^{(i)}$; "back-up" sites meaning those sites which store redundant copies of the data items relevant to *Ti*.

Then we can model the system by a set of trees (forest) as follows: (1) The root of each tree in the forest represents a site $N_f^{(i)} \in N^{(i)}$; and (2) The leaves represent the r redundant copies $n_{d,r}^{i,f}$ of a data item d, $r = 0$ meaning the primary copy. Fig. 2 shows the trees for the example of section 2.1. For instance, site N_1 contains data items x_1, x_2, x_3. Only x_2 and x_3 are relevant to transaction *T*2. x_2 is stored in two other physical copies, besides the one in N_1, while x_3 is stored in just one more copy, and they all can serve as back-up copies.

Such a forest represents a structural model of the transaction *Ti* as to its behavior with respect to failures. The number of leaves $\#L$ is given by

$$\#L = \sum_{j=1}^{\#N^{(i)}} \sum_{i=1}^{D_j} K_i, \qquad (6)$$

where

$\#N^{(i)}$ is the number of primary sites relevant to *Ti*;

D_f is the number of data items, relevant to *Ti*, stored at site N_f;

K_i is the resiliency parameter, i.e. the number of replicas of data item i.

We shall group the data items having the same value of K into the same resiliency class. To avoid unnecessary complexity, in the rest of the paper we shall keep K constant for all the data items in the database, so considering just one resiliency class. This assumption does not affect generality and can be easily removed, allowing a different resiliency parameter for each data item.

Looking at the physical system, however, back-up copies for different data items can be stored at the same site. In a very first approximation, we

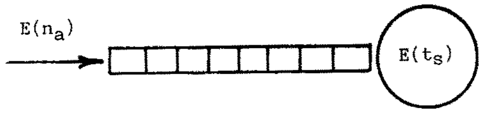


Fig. 3. A service center.

can think that the availability of a data item is determined by the availability of the computing equipment of the site at which it is stored. The rationale for such an assumption lies in the use of a hyerarchical modelling approach, which brings from the availability values of each intervening component (seek arm, disk controller, channel, CPU, etc.) up to the availability values of the whole "access path" to the data item [15]. Therefore we must add a set of constraints of the form:

$$n_{d,r}^{i,f} = n_{e,s}^{i,g} = N_q^{(i)} \in N^{(i)}. \qquad (7)$$

These constraints are shown by the dotted lines in fig. 2, and by the external labels.

The problem having been decomposed in this form, we can build the two vectors giving the static description of the problem.

Let us come now to the failure mechanism. As we said in sect. 3.1, we suppose that failures are determined by a peak overload at one or more sites. The rates at which sites time-out owing to the overload add-up to their hardware failure rates. For the purpose of this paper, we shall consider a site and its workload as a server and its queue.

Let it be (fig. 3): $E(t_s)$ = mean service time; $E(n_a)$ = mean arrival rate; and t_q = response time.

If the queue has exponentially distributed inter-arrival and service times (M/M/1 queue) the probability that the response time exceeds a given time t is [10]:

$$\mathrm{Prob}(t_q < t) = \mathrm{EXP}[-(1-\rho) * t/E(t_s)], \qquad (8)$$

where $\rho = E(n_a) * E(t_s)$ is the utilization factor for the server. The choice of a M/M/1 model is made just for ease of exposition; in fact a M/G/1 model could fit as well, provided we could invert the Laplace transform appearing in the Pollaczek–Khintchine transform equation [7].

Therefore, given a time-out T and a service time $E(t_s)$, the probability that the time-out is exceeded depends exponentially on the utilization factor of the site. Fig. 4 shows the typical relationship between the server utilization factor and the response time in a M/M/1 model. It can be easily seen from it that the higher is the utilization factor ρ_0 of the server, the higher is the probability that even a small overload $\Delta\rho$ will cause the site to fail.

We can evaluate the utilization factor ρ^*, resulting from a rerouting of transactions to back-up data items, in the following way.

Let us define the *repartition coefficient* α as the amount of load each back-up copy has to cope

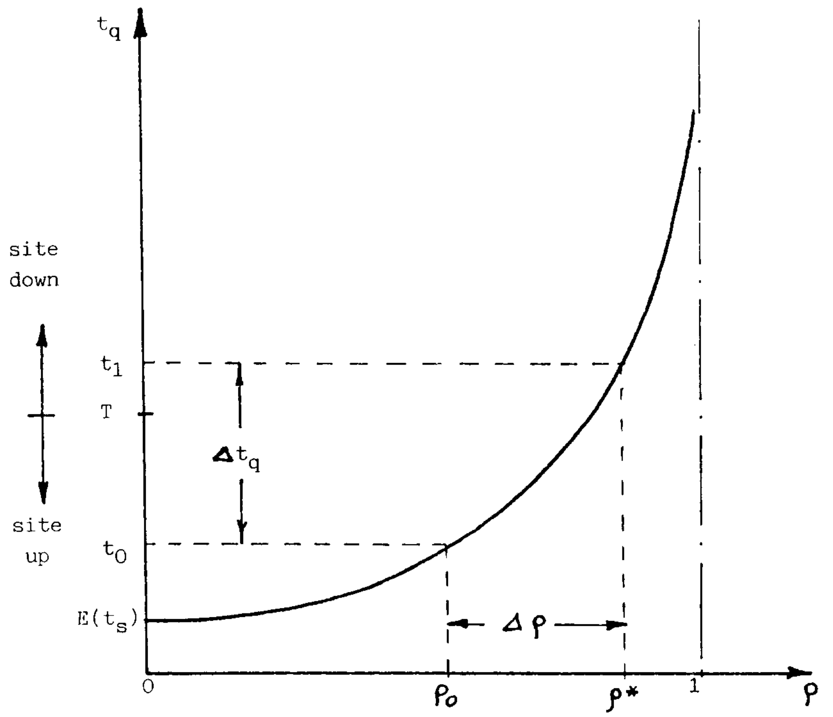


Fig. 4. The workload dependent time-out mechanism.

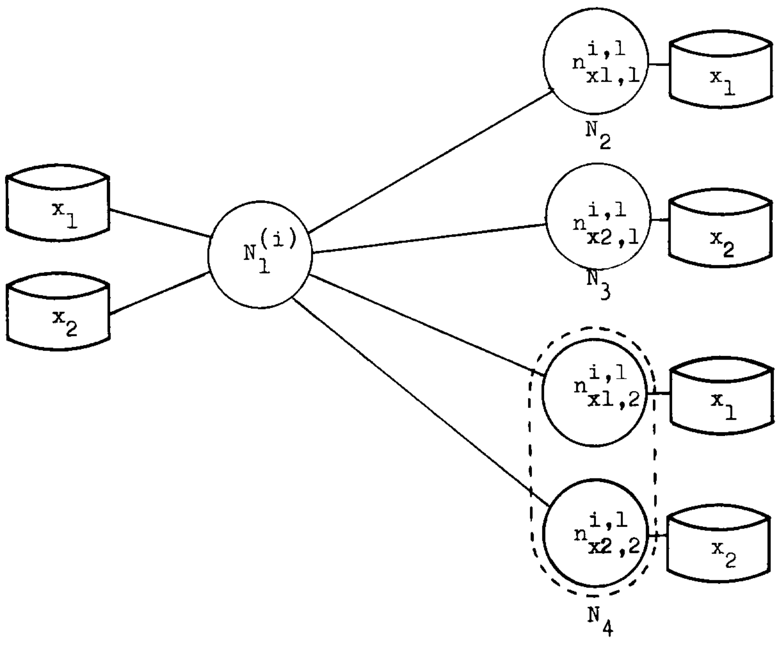


Fig. 5. An example system.

with after the rerouting operation related to its normal production workload. If the system is 1-resilient (i.e. there are two copies of each data item) and the load is balanced between the two copies during normal operation, then $\alpha = 2$. If the system is n-resilient with balanced load and this load is equally subdivided among the n remaining copies in case of failure, then $\alpha = (n+1)/n$. For more complicated conditions of loading and load subdivision, the α's can be given as a table function.

Let it be:

n'_{Np} the mean arrival rate of the requests to a given node Np for all the data items excluding those interested at the rerouting operation;

n_{ij} the mean arrival rate of the requests to the back-up data item j belonging to the Resiliency Class i;

α_i the repartition coefficients for the Resiliency Class i, then

$$\rho^*_{Np}(s) = E\left(n'_{Np} + \sum_i \alpha_i \sum_j n_{ij}\right) * E(t_s). \tag{9}$$

Therefore, given a system as in fig. 5, the probability of having a utilization factor ρ^* for node Np, is equivalent to that of being in a given system state S, and whether in such a state this node is faulty or not depends in turn on the value of $\rho^*(S)$ itself. Then the probability of a given node to become faulty owing to the described mechanism, $\text{Prob}(F_{Np})$, is given by

$$\text{Prob}(F_{Np}) = \text{Prob}(S) * \text{EXP}[-(1-\rho^*(S)) * T/\text{E}(t_s)], \tag{10}$$

where Prob(S) is the probability of being in state S and T is the value of the time-out.

Let us suppose now that the probability of failure of node $N1$ does not depend on the system state (at least through the same mechanism we are

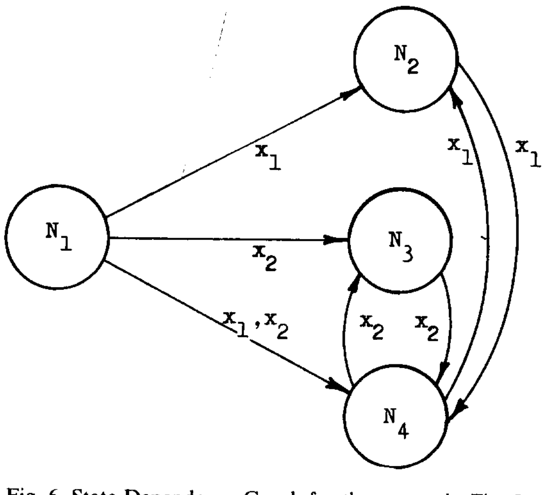


Fig. 6. State Dependency Graph for the system in Fig. 5.

considering), so that we can evaluate it by means of (3). By means of (10) we can then evaluate the probabilities that nodes $N2$, $N3$, $N4$ be faulty and therefore the probability of each resulting state.

The evaluation can be iterated as many times as necessary to exhaust all the possible dependencies. The dependencies themselves can be clearly put into evidence by means of the *state dependency graph*. This graph is so defined:

1. there is one vertex for each physical node Ni.
2. there is an oriented edge between any two nodes $Ni \rightarrow Nj$ such as the failure of Ni affects the reliability behavior of Nj. Edges are labelled with the resources which require the load transfer.

The State Dependency Graph for the example of fig. 5 is given in fig. 6.

Then, if $N2$ enters a faulty state, we shall apply (10) again to find out if $N4$ fails in turn owing to the consecutive failure of $N1$ and $N2$. Failure propagation ends when a dependency path has been completely tested. In fig. 6 we can find paths $\{N1, N2, N4, N3\}$, $\{N1, N3, N4, N2\}$, and $\{N1, N4, [N3, N2]\}$, where the bracketed sets are ordered sets from left to right, while sets included in square brackets represent parallel branches.

Such a procedure exhausts all the dependencies for a given root node in the forest of fig. 2. However, the example of fig. 5 does not consider the fact that back-up nodes can be – in turn – the root nodes for other parts of the given transaction or, – even worse, if the hypothesis of independence of the transactions does not hold – for parts of other transactions, so increasing the propagation effects. This case is described by more complex cyclic graphs, as we shall see in section 3.4.

3.3. Total Transition Probabilities

Since each node Ni has a failure probability $f_{\mathrm{h}}^{(i)}$ due to hardware faults (see section 2.2), the probability of transition from a state Sk, in which Ni is working, to a state Sj, in which it is faulty, is

$$\mathrm{Prob}(Sk \rightarrow Sj) = F\left(f_{\mathrm{h}}^{(i)}, \rho_{Ni}(Sk), T\right). \tag{11}$$

Since the hardware failure and the time-out failure mechanisms can be considered fairly independent, probability theory gives us the expression of the joint probability of one or the other of two events, as follows:

$$f^{(i)}(Sk) = \mathrm{Prob}(Sk \rightarrow Sj)$$
$$= f_{\mathrm{h}}^{(i)} + \mathrm{Prob}(F_{Ni}) - f_{h}^{(i)} * \mathrm{Prob}(F_{Ni}) \tag{12}$$

As to the repair probabilities, they depend on the type of failure and, in general, on the state. However as a first approximation, we can assume that, for a given kind of failure, the repair probability is weakly dependent on the state.

Therefore, given a set of repair probabilities $R^{(i)} \equiv \{r_k^{(i)}\}$, one for each kind of failure component i can undergo, we can evaluate the total repair probability as a linear function, weighting them with the probability γ_k of occurrence of each kind of failure in the following way:

$$r^{(i)} = \sum_{k=1}^{R} \gamma_k r_k^{(i)}, \tag{13}$$

where R is the cardinality of $R^{(i)}$, and $\sum_{k=1}^{R} \gamma_k = 1$.

The probabilities $f^{(i)}(S)$ and $r^{(i)}$ so obtained are the values to be inserted as the a_{ij} entries in the A matrix of (3).

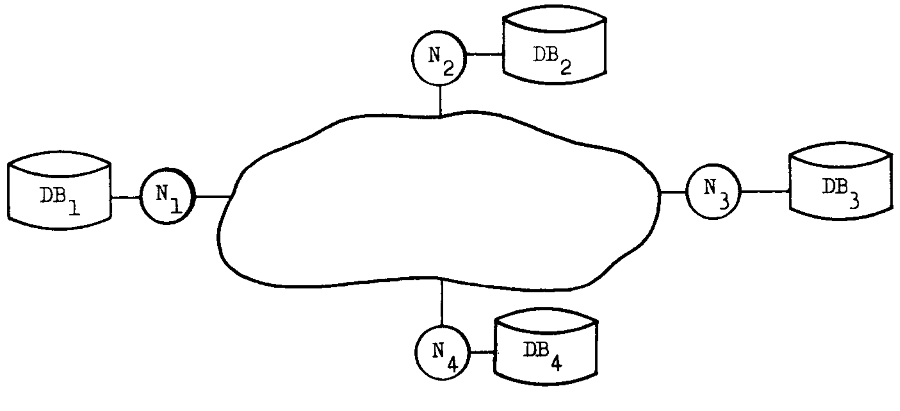


Fig. 7a. An example system.

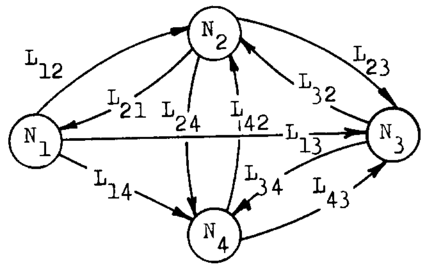


Fig. 7b. State Dependency Graph for the system in Fig. 7a.

3.4. A more complex example

While in the example of fig. 6 there was some kind of hierarchy, since node $N1$ was considered to fail first and the files stored on it were considered to be as primary copies, we can consider a more complex example in which no node is "privileged". Let us look at the system of fig. 7a and let it be described by the State Dependency Graph of fig. 7b. Obviously to pass from one representation to the other we must know all the data items requested by the transactions in the system and their repartition coefficients. However, as far as our computation is concerned, only the loads switched between any two nodes are interesting; we call L_{ij} the load switched from node Ni to node Nj upon the failure of the former.

Fig. 7c shows the four different successions of state dependency graphs which represent the system when, leaving from the situation of fig. 7b, successive failures (one at a time) require load switching among the nodes and the degenerative phenomenon described in sect. 3.1 takes place. Notice that L_{ij} in state Sh generally differs from

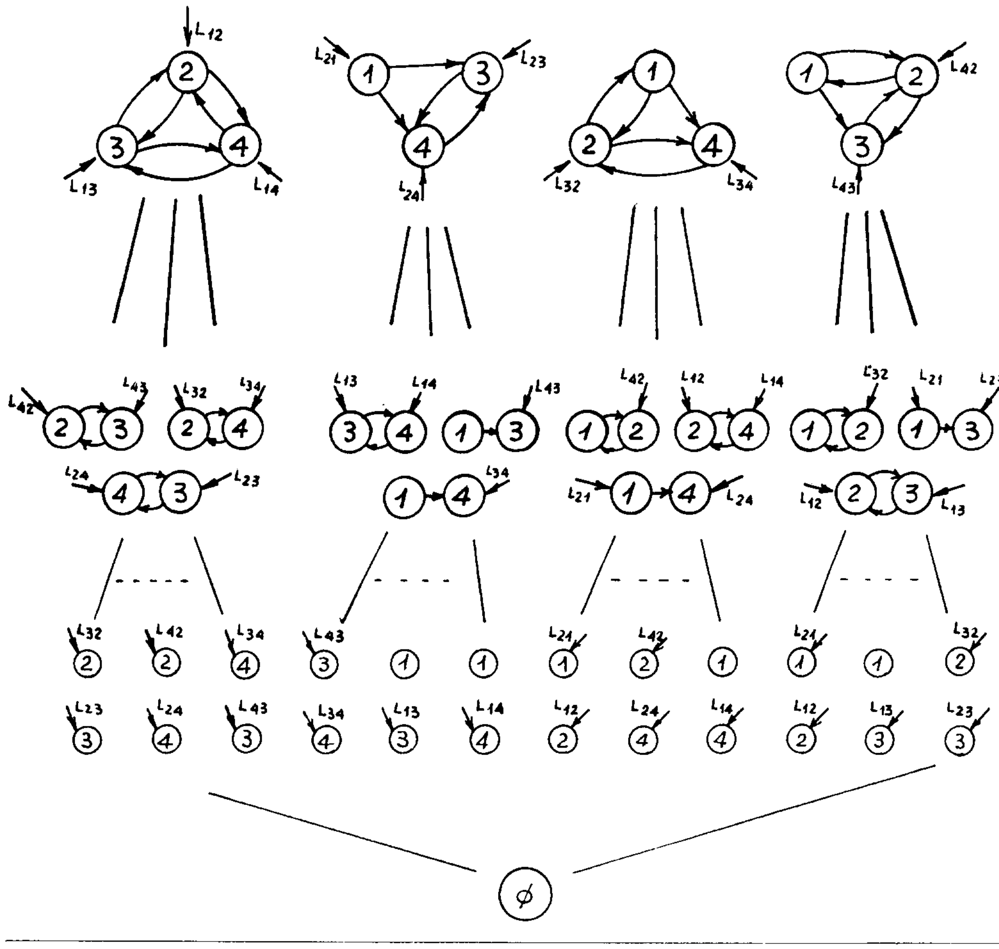


Fig. 7c. The succession of the State Dependency Graphs for the different degeneration paths.

L_{ij} in state Sk owing to the different failure paths followed by the system.

Since the transition matrix in Eq. (3) requires the knowledge of the transition probabilities of each component in each state, we must evaluate relation (12) for all the working nodes in each state. It is easy to see that the total number of times (12) is to be computed is:

$$\sum_{i=1}^{N} N!/(N-i)! \tag{14}$$

The exponential complexity in N of (14) is obviously a limit to the size of the system we can consider. However we saw by (5) that the application of a divide and conquer philosophy is often possible, so that N never reaches values which make the evaluation computationally unpractical.

An actual failure path leaves from the situation of fig. 7b and it proceeds down the tree, choosing one branch at a time, from one level to the other until either it stabilizes itself at one of the system states shown in fig. 7c, or it comes to the complete black-out of the system (state ϕ).

An analytic simulation program is being written on a VAX/780 under Berkeley UNIX which accepts as input the system parameters such as the transition rates, service times, requests arrival rates, time-outs values of each node, and the transaction parameters such as the state dependency graph, and the repartition coefficients, and will give as output the system availability value.

This program will be used as an interactive design tool, to determine systems parameters such as the best allocation for back-up copies, the optimal time-out values, the optimal repartition coefficients.

4. Conclusions

The problem has been considered of obtaining quantitative evaluation of availability in Distributed DB systems. The analysis can be useful in an interactive process to study the effect of different design solutions involving the number and the allocation of duplicate data, the values of the transition rates for the components of the system, and the values for time-outs. It has been shown how state-dependent failures can lead to catastrophic effects if not carefully controlled. The transition probabilities have been obtained for a particular case.

Future work includes a generalization of the methodology presented in this paper and extension to other kinds of state-dependent failures, even if the feeling of the author is that there are few possibilities to arrive to a very general model which encompasses every kind of state-dependent failure.

Acknowledgement

I am greateful to Prof. M. Santomauro and to Ing. P. Gubian for the fruitful discussions we had on this subject.

References

[1] P.A. Alsberg, J.D. Day: A Principle for Resilient Sharing of Distributed Resources. *2nd Int. Conf. on Software Engineering*, Dec. 1976.

[2] V. Amoia, G. De Micheli, M. Santomauro: Computer Oriented Formulation of Transition Rate Matrices Via Kronecker Algebra. *IEEE-Trans. Reliability, 1981.*

[3] R.E. Barlow, F. Proschan: Mathematical Theory of Reliability. John Wiley & Sons, 1965.

[4] J.N. Gray: Notes on Database Operating Systems – in *Operating Systems: an Advanced Course*, Springer Verlag, 1977.

[5] M.W. Hammer, D.W. Shipman: Reliability Mechanisms for SDD-1, a System for Distributed Databases. *ACM-TOS*, Vol. 5, n. 4, Dec. 1980.

[6] K.B. Irani, N.G. Kabbaz: A Model for a Combined Communication Network Design and File Allocation for Distributed Databases. *Proc. 1st Int. Conf. on Distrib. Computing Systems*, Huntsville, Oct. 1979.

[7] L. Kleinrock: Queuing Theory-Vol. 1. *John Wiley & Sons*, 1975.

[8] P. Lancaster: Theory of Matrices. *Academic Press*, 1969.

[9] B.G. Lindsay, P.G. Selinger: Notes on Distributed Databases. Advanced Course on Distributed Databases, Sheffield City Polytechnic, 1979.

[10] J. Martin: System Analysis for Data Transmission. *Prentice-Hall*, 1972.

[11] G. Martella, B. Ronchetti, F.A. Schreiber: Availability Evaluation in Distributed Database Systems. *Performance Evaluation*, Vol. 1, n. 3, North Holland, 1981.

[12] G. Martella, B. Pernici, F.A. Schreiber: Distributed Databases Reliability Analysis and Evaluation. *2nd Symp. on Reliability in Distributed Software and Database Systems*, IEEE-CS/ACM, Pittsburg, July 1982.

[13] B. Randell: System Structure for Software Fault-Tolerance. IEEE-TSE, vol. 1, n. 2, June 1975.

[14] G. Martella, F.A. Schreiber: Improving Access Interference in Distributed databases. *18th Allerton Conference on Communication, Control, and Computing*, Allerton House Monticello (Ill.), Oct. 1980.

[15] G. Martella, B. Pernici, F.A. Schreiber: An availability Model for Distributed Transaction Systems. Internal report, submitted for publication.