Also,

$$E\left[\left(\sum_{i=1}^{R} \frac{\Lambda_i e^{-\Lambda_i p(t+s)}}{1-e^{-\Lambda_i pt}} - \Lambda(t+s)\right)^2\right]$$
$$= \operatorname{Var}\left(\sum_i \frac{\lambda_i e^{-\lambda_i p(t+s)}}{1-e^{-\lambda_i pt}}\, \psi_i(t) + \sum_i \lambda_i \psi_i(t+s)\right)$$
$$= \sum_i \frac{\lambda_i^2 e^{-2\lambda_i p(t+s)}}{1-e^{-\lambda_i pt}}\, e^{-\lambda_i pt}$$
$$+ \sum_i \lambda_i^2 e^{-\lambda_i p(t+s)}(1-e^{-\lambda_i p(t+s)})$$
$$+ 2\sum_i \lambda_i^2 e^{-2\lambda_i p(t+s)}$$
$$= \sum_i \lambda_i^2 e^{-\lambda_i p(t+s)}\left[1+\frac{e^{-\lambda_i p(t+s)}}{1-e^{-\lambda_i pt}}\right].$$

The above estimate presupposes that a bug's failure rate becomes known when the bug is discovered. If this is not the case, then we can still use the data obtained by t to estimate $\Lambda(t+s)$. One approach is to note that

$$\frac{N_i(t)}{t} \approx \lambda_i$$

where $N_i(t)$ is the number of detected errors caused by bug i. Hence, using (3) we can estimate $\Lambda(t+s)$ by

$$\Lambda(t+s) \overset{\text{est}}{=} \frac{1}{t}\sum_i \frac{N_i(t)\, e^{-N_i(t)p(t+s)/t}}{1-e^{-N_i(t)p}}$$
$$= \frac{1}{t}\sum_{i=1}^{\infty} M_i(t)\frac{ie^{-ip(1+s/t)}}{1-e^{-ip}}. \qquad (4)$$

A second approach to estimating $\Lambda(t+s)$ is to note the following:

$$E[\Lambda(t+s)]$$
$$= \sum_i \lambda_i e^{-\lambda_i p(t+s)}$$
$$= \sum_i \lambda_i e^{-\lambda_i pt}\left(1-\lambda_i ps+\frac{(\lambda_i ps)^2}{2}-\frac{(\lambda_i ps)^3}{3!}+\cdots\right)$$
$$= \frac{1}{p}E\left[\sum_{i=1}^{\infty}\frac{M_i(t)}{t^i}\, is^{i-1}(-1)^{i+1}\right].$$

The last equality following since

$$E[M_j(t)] = \sum_i e^{-\lambda_i pt}\frac{(\lambda_i pt)^j}{j!}.$$

Hence, the above suggests the possible estimator

$$\Lambda(t+s) \overset{\text{est}}{=} \frac{1}{p}\sum_{i=1}^{\infty}\frac{M_i(t)}{t^i}\, is^{i-1}(-1)^{i+1}. \qquad (5)$$

Although we intuitively favor (4), numerical tests are needed to see whether (4) or (5) yields the better estimates. Of course, s should not be too large in relation to t for either estimate to be very effective.

We can also use the above to estimate the expected number of new bugs discovered in $(t, t+s)$. As this quantity is equal to $pE[\int_0^s \Lambda(t+y)\,dy]$, it follows from (3) that we can estimate this quantity by

$$\sum_{i=1}^{R}\int_0^s p\,\frac{\Lambda_i e^{-\Lambda_i p(t+s)}}{1-e^{-\Lambda_i pt}}\,dy = \sum_{i=1}^{R}\frac{(1-e^{-\Lambda_i ps})\,e^{-\Lambda_i pt}}{1-e^{-\Lambda_i pt}}.$$

When the failure rates do not become known when a bug is detected, we can estimate the expected number of bugs that will be discovered between t and $t+s$ by

$$\sum_i M_i(t)\frac{e^{-ip}}{1-e^{-ip}}(1-e^{-ips/t}).$$

Remarks: The results in this section can be used to devise an easily implemented stopping rule for testing. One could test for a time t and then, based on the observed data, choose an additional time testing time s such that the estimated error rate at s would be appreciable. One can then reevaluate this after testing for the additional time to determine whether to stop or continue for an additional time indicated by the above.

References

[1] P. Diaconis, unpublished notes on statistical decision theory.
[2] B. Efron and R. Thisted, "How many words did Shakespeare know," *Biometrika*, vol. 63, pp. 435-447, 1976.
[3] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Rel.*, vol. R-28, pp. 206-211, 1979.
[4] Z. Jelinski and P. B. Moranda, "Software reliability research," in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed. New York: Academic, 1972, pp. 485-502.
[5] B. Littlewood, "A critique of the Jelinski-Moranda model for software reliability," in *Proc. Annu. Rel. Maintain. Symp.*, 1981, pp. 357-364.
[6] B. Littlewood and J. L. Verrall, "A Bayesian reliability growth model for computer software," in *Rec. IEEE Symp. Comp. Software Rel.* New York: IEEE, 1973, pp. 70-77.
[7] C. V. Ramamoorthy and F. B. Bastani, "Software reliability-status and perspectives," *IEEE Trans. Software Eng.*, vol. SE-8, July 1982.
[8] H. Robbins, "Estimating the probabilities of unobserved events," *Ann. Math. Stat.*, vol. 39, pp. 256-257, 1968.

An Availability Model for Distributed Transaction Systems

GIANCARLO MARTELLA, BARBARA PERNICI, AND FABIO A. SCHREIBER

***Abstract*—A method is proposed for quantitatively evaluating the availability of a distributed transaction system (DTS). The DTS dynamics can be modeled as a Markov process. The problem of formulating the set of linear homogeneous equations is considered, obtaining the related coefficient matrix, that is, the transition rate matrices of the DTS elements. Such operations can be performed according to the rules of Kronecker algebra. The transition rate matrices are used to calculate the probabilities of the different possible states of the DTS. The**

Manuscript received March 21, 1984; revised September 12, 1984 and February 11, 1985. This work was supported in part by a CNR-PFI Datanet Project and by the Ministero per la Pubblica Istruzione.

G. Martella is with the Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.
B. Pernici is with CSISEI-CNR, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.
F. Schreiber is with the Dipartimento di Matematica, Universita di Parma, Parma, Italy.

availability with respect to a transaction T is computed through its representation by means of a structure graph and a structure vector related to the probabilistic state of the DTS element relevant to the transaction T itself.

Index Terms—**Availability, distributed databases, Markov models, performability, reliability.**

I. Introduction

One of the prime motivations for building distributed transaction systems (DTS) is to enhance availability with respect to what would be provided by a centralized system. The redundancy of data and processors provided by a DTS potentially enables it to continue its work despite the failure of individual sites. The growth in complexity of DTS and their use in ever more critical application areas make their availability an important design consideration.

The problem of availability and reliability evaluation has been already considered by researchers [1]–[3]. Modeling of computer reliability, starting from the earlier works [4], has stressed the representation of the probabilistic nature of structural changes caused by faults in the computer elements. A modified Markov model has been used to evaluate a number of computation related measures for degradable computing systems [5].

An approach to some aspects of reliability and availability evaluation in distributed databases is presented in the following papers: in [6] dependency among fragments of data is studied in relation to their availability; in [7] particular emphasis is put on the structure of the database considering host nodes as fundamental elements; in [20] the workload dependency of transition rates is considered in a real-time distributed database.

A. The Distributed Transaction System Environment

In order to perform a quantitative analysis of availability in a DTS, we must identify the components which concur in determining its behavior.

Fig. 1 shows the scenario of the DTS from a transaction point of view. It is a fairly common assumption that the distribution issues are transparent to the application programs implementing the transaction, i.e., a transaction requiring access to data and processing on different computers (called global transaction) is written as if it were completely performed locally. Therefore, a global transaction will be defined in terms of a set of global data belonging to a distributed database and a set of operations to be performed on them.

For the ease of description and without affecting the generality of the method, the DTS database is considered to be based on a relational model of data. Relations can be either horizontally or vertically partitioned; moreover, in order to have a good performance of the system also in case of failure of one or more nodes, copies of relations as a whole or of fragments of the partitioned relations can be stored at several nodes [8]. When distinction among them is not needed we shall call relations, fragments, and copies by the generic name of data items. All the data items of the DTS are stored in physical files.

The distributed computing system, which is the support of DTS, is made up of two kinds of subsystems: the *host nodes* and the *telecommunication network*. The host nodes are constituted by the cooperating computing centers. The telecommunication network is constituted by transmission and switching devices, linking the host nodes.

Moreover, in a multiuser/multiprogrammed environment, such as found in most of the host nodes, a *concurrency control mechanism* is needed which serializes the execution of transactions accessing the same data at the same time.

On this system a set $\{T\}$ of transactions is processed. Each transaction $\underline{T}$ requires a number of *data items* (files) to be

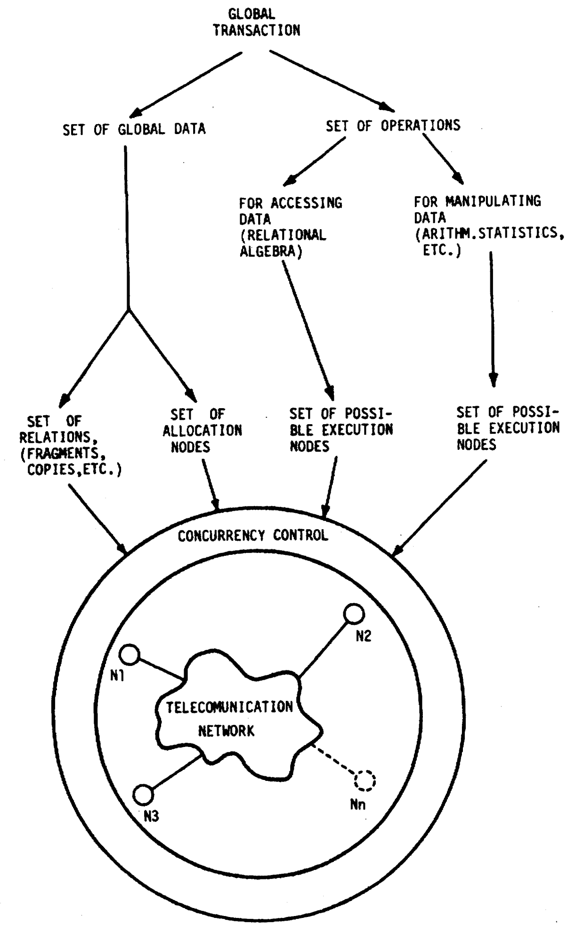

GLOBAL TRANSACTION
SET OF GLOBAL DATA
SET OF OPERATIONS
FOR ACCESSING DATA (RELATIONAL ALGEBRA)
FOR MANIPULATING DATA (ARITHM.STATISTICS, ETC.)
SET OF RELATIONS, (FRAGMENTS, COPIES,ETC.)
SET OF ALLOCATION NODES
SET OF POSSIBLE EXECUTION NODES
SET OF POSSIBLE EXECUTION NODES
CONCURRENCY CONTROL
N1
N2
N3
Nn
TELECOMUNICATION NETWORK

Fig. 1. The distributed transaction system "scenario."

available in order to produce the desired results. Task of the system is to determine the distribution features of the global data relevant to the transaction, such as the logical fragmentation of relations and/or their duplication, and the set of all the allocation nodes where they are physically stored; moreover, a set of all the execution nodes must be determined, where the operations, related to the transaction itself, can be performed.

We distinguish two data items sets for a given transaction $\underline{T}$: the read-set $\underline{RS}_T$, composed of the data items that must be read in order to process the transaction, and the write-set $\underline{WS}_T$, composed of the data items that must be updated by the transaction itself. We assume that, to successfully complete a transaction, at least one copy of every element of the read-set and at least k copies (if we want k-resiliency in updates [1], [6], [7]) of every element of the write-set have to be available.

The topmost level of performance of a DTS system is attained when all the transactions belonging to $\{T\}$ can be processed. In this paper we define the capability of execution of the most critical transaction alone as the least acceptable accomplishment level. Therefore, in the following, we are going to evaluate the availability of the system with respect to such a transaction.

Then, the system is available (i.e., the transaction can be processed) if and only if at least one of the possible copies of each of the data items relevant to the transaction is available, all of the execution nodes which are unique resources for the transaction and at least one of those which are considered to be hot backup are available, and at least one communication path exists between the nodes which must exchange information, the node issuing the transaction included.

Mastering a system of such a complexity, requires a "divide and conquer" philosophy. Therefore, we identify several levels of description, each one composed of many subsystems; each subsystem can be modeled (as far as availability is concerned) in terms of its transition rates between the working and the failed state and vice versa. Transition rates for subsystems at one level can be, in principle, evaluated from the transition rates of the components of the more detailed models at the lower levels [29].

The description level of DTS identified in this paper is presented in Section II, where the availability model assumptions are also discussed. In Section III a two-step method is proposed for availability evaluation.

II. The Approach

A. DTS Elements

We will evaluate the DTS availability starting from a level whose subsystems are as follows:

- data-items
- concurrency control policies
- host nodes
- communication paths.

The behavior of each subsystem is represented by its transition rates between the working and the failed state and vice versa. In a DTS, the rates may belong to one of the following classes:

independent rates, i.e., their value does not depend on the state of any other component in the system, nor on the workload on it;

state-dependent rates, i.e., their value changes depending on the state (working or failed) of some other component in the DTS, be it of the same or of different type;

workload-dependent rates, i.e., their value changes depending on the workload applied to the component itself.

Our model can deal with both independent and dependent rates. In particular, state dependence is accounted for through a stress effect caused on a component i by the failure of component r.

The stress effect is represented by a *stress adjustment factor* γ_{ir}

$$\gamma_{ir} = \frac{\lambda_{i_{x_r=0 \wedge x_j=1: \forall j \neq r}}}{\lambda_{i_{x_j=1: \forall j}}}. \tag{2.1}$$

If there is no dependence between two components, then $\gamma_{ir} = 1$.

The stress effect on component i due to the failure of a set of components can be represented as the chain product of the stress adjustment factors of component i pertinent to all failed components (chain s-dependence assumption)

$$\gamma_{ir_1r_2\cdots r_k} = \gamma_{ir_1} * \gamma_{ir_2} * \cdots * \gamma_{ir_k}. \tag{2.2}$$

This assumptions is a simplification of the real context since only direct dependencies between components can be considered, but it substantially reduces the amount of data required in the computation of availability.

The actual evaluation of the behavior of the transition rates and of the value of the stress adjustment factors is mainly based on the statistical analysis of historical series of values, collected in already working systems [14]; however, in some simple cases, analytical and simulation modeling can prove useful to this purpose [20].

Moreover, the most general case of state-dependent transition rates claims for an all-from-all dependence. Such a situation would pose very hard computation problems, owing to the exponential nature of the related algorithms [7].

For these reasons, as a first approximation, the independence hypothesis is usually applied. In this paper, we shall use state-dependent rates for the data items, while we shall keep the independence hypothesis for the other subsystems. Anyhow, for every subsystem we shall show some dependence issues, which are still an open research field, and the approximations which are to be made in order to consider these subsystems as if they were state/workload independent.

TABLE I
Symbol Definition

D^i	data items on node N_i
d_i	data item (D: number of data items)
i	element of the DTS (n: number of elements)
l_{ij}	virtual link from node i to node j (L: number of links)
N_i	node (N: number of nodes)
RS_T	read-set of T
s_i	semaphore (S: number of semaphores)
$\{T\}$	set of transactions on DTS
T	transaction
WS_T	write-set of T
$A_T(t)$	availability with respect to T
c_T	structure vector
$p(t)$	probability vector
Q	state space of the DTS system
q	value in Q
Q_i	subspace of the DTS system
$R_T(t)$	reliability with respect to T
t	time
$u(t)$	state trajectory in $Q * \tau$
x_i	state of element i
X_t	DTS state at t
γ_{ir}	stress adjustment factor for element i due to the failure of element r
Δt	time interval
λ_i	failure rate of element i
μ_i	repair rate of element i
τ	mission period

In Table I all the symbols used in the paper are listed.

Data-Items: Transitions from state X_t to another state $X_{t+\Delta t}$ due to data item failures or repairs are caused by the following:

1) hardware failures and repairs of I/O devices, or disks, magtapes, and so on;

2) system and software failures and repairs such as inconsistency of data, depending, for instance, on incorrect data entry, etc.

Therefore, we assume that the data item is available if the related I/O device is functioning *and* the stored data are in a consistent state.

Actual hardware and software transition rates for individual data items can be obtained by statistical analysis of historical series collected in working installations. Analytical models can prove useful also; in [30] the checkpoint-rollback-recovery mechanism to provide a fault-tolerant operation of transaction systems, by assuring the integrity of the database, is modeled as a continuous-time Markov chain. However, we are going to show that a kind of dependence (state dependence) exists among the transition rates of the data items d_i. The kind of dependence is different for duplicated, fragmented, or simple relations.

Duplicated Relations or Fragments: In the case of a duplicated relation (or fragment), all the operations on a failed copy are transferred to a copy of the relation (or of the fragment) which is still available. This means that the failure rate of each still available copy is increased, both owing to the heavier load on the disk unit and the enhanced danger to data integrity; that is, the stress adjustment factor is greater than one. The actual value of the stress factor can be evaluated on the basis of the manufacturer's reliability data, supposing that the predominant failure mechanism is due to the mechanic stress to the arm, connected to the access frequency. In this case the chain s-dependence assumption can be considered fairly valid: if more than one copy fails, and the load is evenly distributed, the λ_i's of the other copies are proportionally increased. A similar situation, but with a different workload-dependent failure mechanism, is described in [20] for highly available redundant real-time distributed databases.

Fragments: For ease of description, we consider only one level partitioning for a relation; that is, if a relation is partitioned, it can be partitioned only in one way, either horizontally or vertically. Fragments of a partitioned relation exhibit a state-dependent behavior if a transaction exists requiring the relation as a whole. Fragments obtained from a relation via horizontal or vertical partitioning are dealt with in the same way. We suppose that a control mechanism (transaction analysis) exists that prevents the execution of transactions involving a whole relation if one of its parts is unavailable. From these considerations it comes that the failure of a fragment of a relation involves the abortion of the global transaction requiring it, and therefore a lower number of accesses are done on the working fragments, which are still alive for local transactions. Then the probability of a failure changes for each part of the relation if one or more than one of its parts fails. In this case the values γ_{ir} are less than one.

Single Relations: Two relations are considered dependent on each other if at least a transaction exists which requires both relations for its execution. The failure of one of the two relations involves that transaction execution cannot take place, provided the same control mechanism exists as described for fragments. It comes that the probability of failure of a relation is lowered if other relations are not available, that is, the stress factor is less than one, since fewer transactions are done on it.

Concurrency Control Policies: Availability and transition rates of a tangible component (hardware or software) are easily understandable concepts, but it is harder to model the same quantities for the algorithms controlling the DTS, while they greatly influence the availability properties of the whole system.

In this paper we take into account concurrency control for database access, modeling a two-phase locking protocol in DTS [9], [31].

In our model locks are done at data item level; data items are locked to prevent inconsistency in update operations. Semaphores take into consideration that a data item can be unavailable due to the fact that it has been locked by another transaction. The probability of a semaphore to become red (the corresponding data item is locked) or green (the corresponding data item is accessible) depends on the adopted locking policy. Effects of granularity of locked elements and of locking policies on the performances of locking algorithms have been examined in [9].

The computation of the transition rates depends also on transaction frequency and frequencies of access to the different data items. We suppose that if a transaction cannot lock all the data items it needs, it finds the system is in an unavailable state and it is not queued.

From the above considerations failure rates for semaphores can be computed analytically or by simulation [10]. The repair rate μ_i depends on the mean time of execution of a transaction, considering that all locks are not released until transaction termination. In the case of uniform access to the various data items in the database, semaphore transition rates are independent from the state of other semaphores [11]. These rates, however, depend in a complex way from the state of other DTS elements; for instance, the duration of the blocked state for a data item depends on the state of the disk unit. However, semaphore transition rates are considered in our model independent from the state of other DTS elements on the basis of two considerations: first, semaphores working in a perfectly functioning system have the highest possible failure rates, so the consideration of these rates refers to a worst case analysis; second, quantification of state-dependent rates is possible today only through simulation of each particular case and, with this method, state dependency is implied in the mean values resulting from the simulation experiments.

Host Nodes: As shown in Section I, the DTS runs on N host nodes N_i which can perform two types of functions in transaction execution. A node can be an execution node or an allocation node. In both cases the node has to be working in order to perform its function in transaction execution.

Node failures can be due either to internal failures, such as hardware or system software errors, or to external failures, due to the external environment.

Each host node component has its failure and repair rate. However, in our model it is important to know the failure and repair rates of the node on the whole. These values can be obtained from computer producers, from statistical data, from simulation or using one of the analytical models available in the literature [12], [13].

Some works [14] and [15] have outlined a dependence of transition rates values of the CPU from system workload. This workload depends in DTS on the strategy chosen in the execution of single transactions. However, the workload generated in a general purpose multiprogrammed host node by a single transaction is a small fraction of the total workload on the node. The factors that can influence the workload of a host node are contrasting. Host nodes can be both execution and allocation nodes for many transactions. If a node is used as an execution node the stress factor γ for all the possible alternative execution nodes is greater than one. If a node is used as an allocation node, the stress factor γ is greater than one for nodes containing copies of a failed data item and smaller than one for nodes containing nonreplicated fragments or relations.

These considerations suggest to consider, as failure rates, the failure rates of the nodes at a high workload, and to consider, as a consequence, all nodes independent from each other, using a worst case policy in DTS availability evaluation.

Communication Paths: Let us call *virtual links* l_{ij} between two host nodes i and j, the possibility of communication between the two nodes, the communication being possible through whatever physical path between them. The availability of the virtual link implies that at least one of the physical communication paths is available (i.e., it is constituted by functioning elements). Each physical link has a failure and a repair rate, and the topology of the network conditions the dependence of the transition rates of virtual links on the transition rates of physical links constituting it.

In the DTS model we are interested only in transition rates of virtual links, which are for an N nodes system $N * (N - 1)/2$ (the network is logically a complete graph).

Methods for the computation of probabilities of interruptions of a virtual link between two given nodes in a communication network are found in the literature [16]-[18].

The main problem in this field concerns the consideration of state dependence between failure rates of physical links [19].

Virtual links failure rates could be considered state-dependent due to the fact that failures of nodes in the DTS can cause different executions of the transactions, increasing the workload of some of the virtual links; however, the influence of workload on transition failures for virtual links is less important than other factors that can cause physical link dependence and nevertheless are not considered in the above-mentioned network reliability models. Therefore, in the following we will assume the virtual links failure rates independent from each other.

The repair rate μ of a virtual link is obtained in a similar way from the mean time to repair, assuming that multiple failures are repaired contemporarily and independently.

Therefore, the DTS elements are as follows:

N state-independent host nodes $\underline{N}_i$.

L state-independent host node-to-host node virtual links $\underline{l}_{ij}$.

S state-independent semaphores s_i.
D state-dependent data-items d_i.
Then the number n of DTS elements is

$$n = N + L + S + D.$$

B. Availability Model Assumptions

DTS as described are gracefully degradable systems. Failures can be both hardware failures, such as the absence of a virtual link between two nodes or a broken disk, and software failures, such as inconsistency of data, time-outs (possibly caused by overloads of nodes [20]), and so on. In this paper the mechanism to account for both hardware and software failures is the same: each element i of the system can either be available (working state: $x_i = 1$) or not available (failed state: $x_i = 0$). Failure of an element i is the transition from $x_i = 1$ to $x_i = 0$. Repair of an element is the transition from $x_i = 0$ to $x_i = 1$. Each element i of the DTS system has a failure rate λ_i and a repair rate μ_i.

$\lambda_i \Delta t$ and $\mu_i \Delta t$ are, respectively, the probability of a failure and of a repair in the time interval Δt for element i, these expressions holding for small Δt.

We suppose to observe the system in a time interval τ, called the mission period. The mission period is continuous and it is determined by the particular application (i.e., one day, one month, etc.). The behavior of the DTS is formally viewed as a stochastic process. X_t is a random variable taking values in the state-space Q of the DTS. The state-space Q is discrete, since the state X_t of a DTS composed of n elements, with two possible states each, can assume 2^n different values q. A value q can be represented by its equivalent representation in a binary code, where each bit represents the state of one element, elements being ordered in a predefined way; that is if x_i are the states of the n elements, q can be obtained with the following relation:

$$q = (2^n - 1) - \sum_{i=1,n} x_i * 2^{n-i}. \tag{2.3}$$

In our approach it is possible to see the system at different levels of detail. Each element i can be considered as an aggregate of elements of an increased detail. This is equivalent to decompose the state-space Q into a product of subspaces Q_i:

$$Q = Q_1 * Q_2 * \cdots * Q_n \tag{2.4}$$

where Q_i is the subspace of the ith aggregate of elements.

The system behavior in τ is described by its state trajectory $u(t)$, which describes the succession of the states occupied by the system during its mission period.

The probability vector $\boldsymbol{p}(t)$ is composed of the 2^n elements $p_q(t)$, denoting the probability of "$X_t = q$" as a function of t within the mission period τ. The state of the system depends on the failures and repairs of each of its elements. We assume that failure and repair rates are independent of time t.

We suppose, moreover, that no more than one element can either fail or be repaired in the time interval Δt, so that transitions between states of the DTS can be only due to the failure (or repair) of a single element. This consideration allows us to build a Markovian model of the DTS [21], [22].

Let us define the following.

Availability of the DTS with respect to a transaction T: $A_T(t)$ = probability that the system performs $T \in \{T\}$ successfully within the mission period, where maintenance (hardware and software) is performed on the DTS and transactions are executed concurrently.

In the next section the method to compute $A_T(t)$ is introduced.

III. The Method

The aim of the proposed method is to provide a mean of numerically quantifying the fault-tolerance of a DTS. Its main characteristics are the following:

- hardware and software failures are dealt with in the same way
- it can be automatized.

The method consists in the following three phases.

1) Dynamic phase: computation of $\boldsymbol{p}(t)$.

2) Static phase: computation of $\boldsymbol{c}_T$, which represents transaction T in the DTS structure. The elements c_q set to zero indicate that a failure critical to transaction T occurred in state $X_t = q$ in the system. A critical failure for a given transaction occurred in state X_t of the DTS if the transaction cannot be executed in that DTS state.

3) Availability computation.

We assume that the system is always started in a completely functioning condition ($X_0 = 0$, $\boldsymbol{p}(0)' = |1000 \cdots 0|$).[1] This hypothesis corresponds to the real case of global maintenance done periodically (i.e., once a day, once a week) in most working systems. We suppose that after maintenance all the DTS elements are perfectly working, and the shared resources are free.

A. First Phase: Dynamic Phase

The goal of the dynamic phase is to express probabilistically the state of the system at time t through the probability vector $\boldsymbol{p}(t)$ [23], [28].

Let $p_1^{(i)}(t)$ and $p_0^{(i)}(t)$ be the probabilities for element i to be, respectively, in state $x_i = 1$ and $x_i = 0$ at time t.

Starting from state $x_i = 1$ at t, the probability that at $(t + \Delta t)$ element i is in state 0 is equal to $\lambda_i \Delta t$, and in state $x_i = 1$ equal to $(1 - \lambda_i \Delta t)$, provided Δt is small enough to make higher order terms negligible. The sum of the two probabilities is 1. Starting from state $x_i = 0$ at t, the probability that at $(t + \Delta t)$ element i is in state 1 is equal to $\mu_i \Delta t$, and in state 0 equal to $(1 - \mu_i \Delta t)$. We can write the probability of being in state 1 or 0 at time $(t + \Delta t)$ in the following way:

$$p_1^{(i)}(t + \Delta t) = (1 - \lambda_i \Delta t) * p_1^{(i)}(t) + \mu_i \Delta t * p_0^{(i)}(t)$$
$$p_0^{(i)}(t + \Delta t) = \lambda_i \Delta t * p_1^{(i)}(t) + (1 - \mu_i \Delta t) * p_0^{(i)}(t).$$

From these formulas we obtain, for $\Delta t \to 0$, and defining

$$\boldsymbol{p}^{(i)}(t) = \begin{vmatrix} p_1^{(i)}(t) \\ p_0^{(i)}(t) \end{vmatrix} \quad \text{and} \quad \Lambda^{(i)} = \begin{vmatrix} -\lambda_i & \mu_i \\ \lambda_i & -\mu_i \end{vmatrix}$$

$$\dot{\boldsymbol{p}}^{(i)}(t) = \Lambda^{(i)} * \boldsymbol{p}^{(i)}(t) \quad \text{with} \quad \boldsymbol{p}^{(i)}(0)' = |1 \quad 0|. \tag{3.1}$$

$\Lambda^{(i)}$ is called the *transition rate matrix* of element i.

Formula (3.1) can be used to compute the probability vector $\boldsymbol{p}(t)$ of the whole DTS [24], [25].

In fact, if we have a DTS transition rate matrix, Λ_n, we can compute the probability of the DTS of being in each of its states as follows:

$$\dot{\boldsymbol{p}}(t) = \Lambda_n * \boldsymbol{p}(t) \quad \text{with} \quad \boldsymbol{p}(0)' = |1 \quad 0 \cdots 0|. \tag{3.2}$$

Note that the nondiagonal elements of matrix Λ_n, l_{vw}, indicase the probability $l_{vw} \Delta t$ (for a small Δt) of the transition from state w to state v. Columns w in the matrix represent present states, rows v, next possible states. The diagonal elements are the negated sum of nondiagonal elements for each column, representing that from each state there is either a transition to another state or the permanence in that state. With a Markovian model of the DTS we assume that in a cer-

[1] The symbol ′ denotes the transposition of a vector.

tain state $\underline{w}$ only one of the DTS elements that are available can become unavailable in Δt.

Λ_n can be obtained from the $\Lambda^{(i)}$'s of the single elements, as we are going to show in the following section.

Once Λ_n has been obtained, the system of differential equations (3.2) can be automatically solved by a computer. The solution of system (3.2) is the probability vector $\boldsymbol{p}(t)$.

1) DTS Transition Rate Matrix Λ_n: In Section II we considered the transition rates for each virtual link, each semaphore, and each host node independent from the state of any other DTS element (state independence), while we consider that data items can depend on the state of some other data items, according to the transaction set $\{T\}$. Therefore, the computation of the transition rate matrix Λ_n is subdivided into two steps: one calculates the total transition rate matrix Λ_{ind} for state-independent elements, such as virtual links, semaphores and host nodes; the other calculates the total transition rate matrix Λ_{dep} for state-dependent elements such as data items.

Computation of Λ_{ind}: In general, if two DTS elements are independent, the transition rate matrix Λ_2 for the part of the DTS constituted of these two elements can be computed as follows, using Kronecker algebra [25], [26].

For two DTS elements, 1 and 2,

$$\Lambda_2 = \Lambda^{(1)} \oplus \Lambda^{(2)}. \tag{3.3}$$

Element $\underline{j}$ is added to the system formed by $(j - 1)$ elements $(\Lambda_1 = \Lambda^{(1)})$ by means of the Kronecker sum

$$\Lambda_j = \Lambda_{j-1} \oplus \Lambda^{(j)} \tag{3.4}$$

where Λ_j is the transition rate matrix of a DTS with $\underline{j}$ elements, and $\Lambda^{(j)}$ is the transition rate matrix for element $\underline{j}$. The number of possible system states is doubled and the possible transitions between states due to the failure of last introduced element j are indicated putting in the matrix Λ_j $(2^j \times 2^j)$ the element transition rates λ_j and μ_j in the proper rows and columns.

In general, if there are k independent DTS elements, the transition rate of the DTS can be obtained via the formula

$$\Lambda_{\text{ind}} = \Sigma_{i=1,k} \oplus \Lambda^{(i)}. \tag{3.5}$$

Computation of Λ_{dep}: Let us decompose each transition rate matrix $\Lambda^{(i)}$ for element $\underline{i}$ in the following way:

$$\Lambda^{(i)} = L^{(i)} + M^{(i)} \tag{3.6}$$

where

$$L^{(i)} = \begin{vmatrix} -\lambda_i & 0 \\ \lambda_i & 0 \end{vmatrix} \quad \text{and} \quad M^{(i)} = \begin{vmatrix} 0 & -\mu_i \\ 0 & \mu_i \end{vmatrix}.$$

The DTS transition rate matrix for k-dependent elements, Λ_k is the sum of a failure part L_k, and a maintenance part M_k

$$\Lambda_{\text{dep}} = \Lambda_k = L_k + M_k. \tag{3.7}$$

The failure part L_k considers the state dependence of failures, while M_k considers the maintenance, as performed independently from the data item states.

Let

$$G_{ir} = \begin{vmatrix} 1 & 0 \\ 0 & \gamma_{ir} \end{vmatrix}$$

be the *stress adjustment matrix* of data item $\underline{i}$ due to the failure of the data item $\underline{r}$. L_k is obtained as follows:

$$L_k = \Sigma_{i=1,k} [(\Pi_{r=1,i-1} \otimes G_{ir}) \otimes L^{(i)} \otimes (\Pi_{r=i+1,k} \otimes G_{ir})]. \tag{3.8}$$

The proof is straightforward and can be found in [32].

M_k is obtained as follows (see also formula (3.5) for the state-independent case)

$$M_k = \Sigma_{i=1,k} \oplus M^{(i)}. \tag{3.9}$$

Computation of Λ_n: Λ_n is computed from Λ_{ind} and Λ_{dep} using the formula for the state-independent case

$$\Lambda_n = \Lambda_{\text{ind}} \oplus \Lambda_{\text{dep}}. \tag{3.10}$$

B. Second Phase: Static Phase

The static phase consists of the construction of a model of the transaction in the DTS; this model is constituted by the structure graph and the structure vector [28].

1) Structure Graph for a Transaction: In order to compute availability, we need to know, besides the probabilistic state of the DTS, which elements of the DTS are relevant to the execution of the considered transaction $\underline{T}$.

For ease of exposition, but without a loss of generality, we shall consider henceforth that the set of execution nodes coincides with the necessary allocation nodes.

The condition which must be met for a successful completion of the read access of $\underline{T}$ is that each relation contained in $\underline{RS_T}$ must be available. Relations can be partitioned or duplicated; availability of a duplicated relation means that *at least* one copy must be available and availability of a partitioned relation means that all the fragments of the partitioned relation must be available. The completion of a write access, on the contrary, requires the availability of a number of copies of duplicated relations which depends on the resiliency degree.

Each transaction starts from a node $\underline{N}_i$. We suppose that all the data items must be reachable from node $\underline{N}_i$ at the nodes where they are stored. We do not consider optimization in accessing distributed relations, since this is not in the aim of the paper.

These conditions can be visualized on a directed acyclic graph, $G = \{V, E\}$, called the structure graph.

The elements of $\underline{G}$ are as follows:

- one vertex, called source S, without incoming edges, which represents the origin of the transaction
- one vertex, called drain D, without outgoing edges, which represents the committing of the transaction
- edges E, which represent the DTS elements that are relevant to transaction $\underline{T}$.
- vertices V, connecting the edges from S to D

Edges are connected in series if the elements they represent are all necessary for performing the transaction.

Each read-set is formed by the serial connection of the graph parts relative to each relation.

Graph $\underline{G}$ is formed by the serial connection of the graph parts relative to the edge corresponding to the starting node, the subgraph relevant to the read-set and the subgraph relevant to the write-set, if the write-set is not empty.

K-resiliency is represented in the structure graph with $\binom{m}{k}$ parallel groups of $\underline{k}$ serial edges. The simplest and most usual case is the one in which at least one copy must be updated before the transaction commitment (1-resiliency), with $\underline{m}$ strategies ($\underline{m}$ parallel edges).

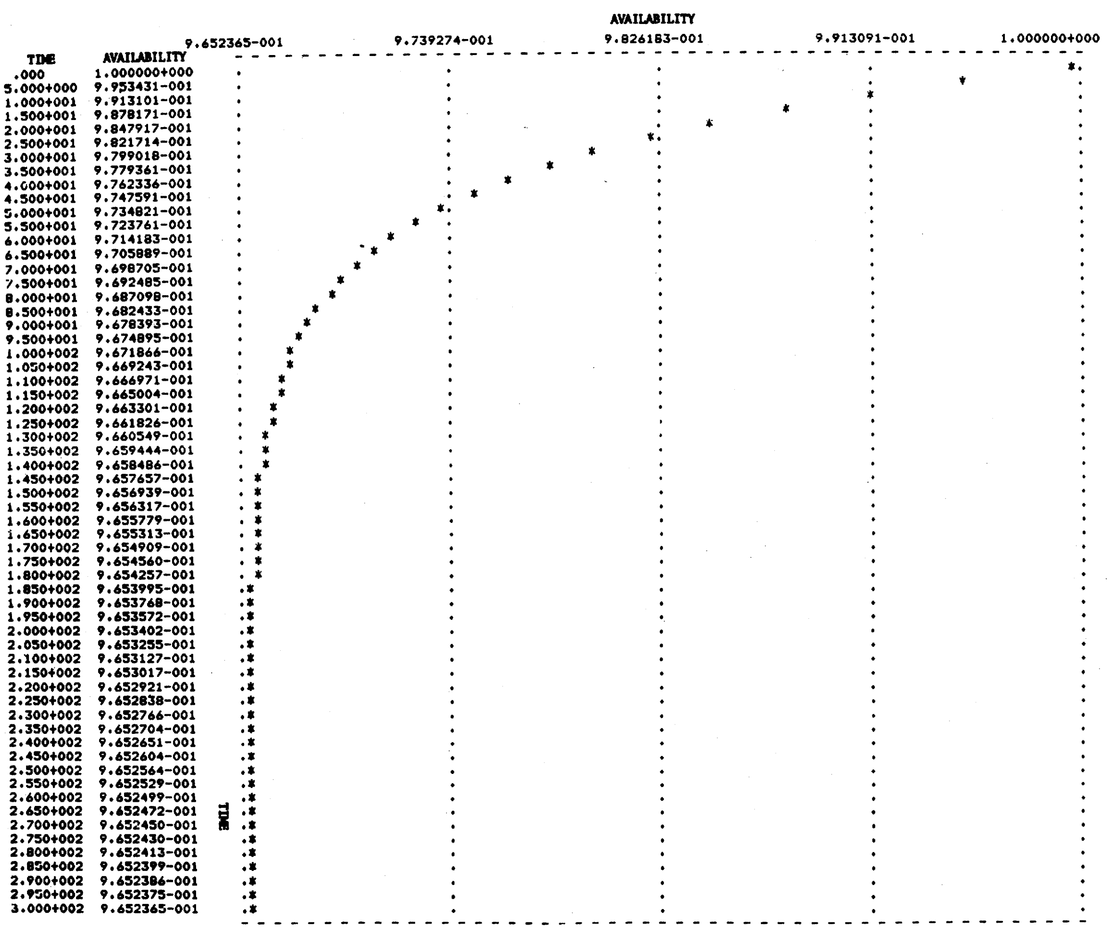
Transaction $\underline{T}$ can be executed if there is at least one path from the source to the drain where every element is in a working state. The path is not representative, however, of the real order of access to the different DTS database relations and the relative DTS elements. This order depends on the chosen execution strategy, and it is not a matter of availability of the system. All we need is that the structure graph be at least 1-connected.

2) The Structure Vector: The structure graph can be associated with a Boolean expression, the Structure Graph Logical Expression SGLE, that inidcates whether in DTS state X_t the transaction can be executed or not.

The steps to obtain the SGLE are the following.

- To each edge the state $\underline{x}_i$ of the corresponding element is associated.
- Serial elements are connected with an .AND. operator.
- Elements in parallel are connected with an .OR. operator.

A group of two or more elements connected in parallel is closed in brackets.



```
   TIME      AVAILABILITY
   .000      1.000000+000
 5.000+000   9.953431-001
 1.000+001   9.913101-001
 1.500+001   9.878171-001
 2.000+001   9.847917-001
 2.500+001   9.821714-001
 3.000+001   9.799018-001
 3.500+001   9.779361-001
 4.000+001   9.762336-001
 4.500+001   9.747591-001
 5.000+001   9.734821-001
 5.500+001   9.723761-001
 6.000+001   9.714183-001
 6.500+001   9.705889-001
 7.000+001   9.698705-001
 7.500+001   9.692485-001
 8.000+001   9.687098-001
 8.500+001   9.682433-001
 9.000+001   9.678393-001
 9.500+001   9.674895-001
 1.000+002   9.671866-001
 1.050+002   9.669243-001
 1.100+002   9.666971-001
 1.150+002   9.665004-001
 1.200+002   9.663301-001
 1.250+002   9.661826-001
 1.300+002   9.660549-001
 1.350+002   9.659444-001
 1.400+002   9.658486-001
 1.450+002   9.657657-001
 1.500+002   9.656939-001
 1.550+002   9.656317-001
 1.600+002   9.655779-001
 1.650+002   9.655313-001
 1.700+002   9.654909-001
 1.750+002   9.654560-001
 1.800+002   9.654257-001
 1.850+002   9.653995-001
 1.900+002   9.653768-001
 1.950+002   9.653572-001
 2.000+002   9.653402-001
 2.050+002   9.653255-001
 2.100+002   9.653127-001
 2.150+002   9.653017-001
 2.200+002   9.652921-001
 2.250+002   9.652838-001
 2.300+002   9.652766-001
 2.350+002   9.652704-001
 2.400+002   9.652651-001
 2.450+002   9.652604-001
 2.500+002   9.652564-001
 2.550+002   9.652529-001
 2.600+002   9.652499-001
 2.650+002   9.652472-001
 2.700+002   9.652450-001
 2.750+002   9.652430-001
 2.800+002   9.652413-001
 2.850+002   9.652399-001
 2.900+002   9.652386-001
 2.950+002   9.652375-001
 3.000+002   9.652365-001
```

Fig. 2. Availability computation.

It is possible to simplify the SGLE by applying the fundamental theorems of Boolean algebra.

This expression is computed for every possible state of the DTS system. If the SGLE, computed for a DTS state $X_t = q$, takes value 0, a failure critical to that transaction has occurred in that state.

The structure vector c_T is obtained as follows: the value computed with the SGLE for state $X_t = q$ is assigned to element c_q. Elements c_q set to 1 in the structure vector thus represent the states in which it is possible to execute transaction $\underline{T}$.

Given:

- the logical and physical distributions of the relations on the different nodes,
- the starting node of the transaction,
- RS_T,
- WS_T,
- the resiliency parameter k,

the SGLE can be derived automatically [7], [27] (Appendix I).

C. Third Phase: Availability Computation

DTS availability is computed using the probability vector $p(t)$ and the structure vector c_T

$$A_T(t) = c_T' * p(t). \tag{3.11}$$

Some numerical examples have been presented, showing the practical use of this method in the case of a distributed database system with duplicated files and state independence among the transition rates [7], and of a fragmented database with state dependence among the transition rates of the fragments belonging to the same file [6]. We show some of these results in Appendix II.

IV. Summary and Conclusions

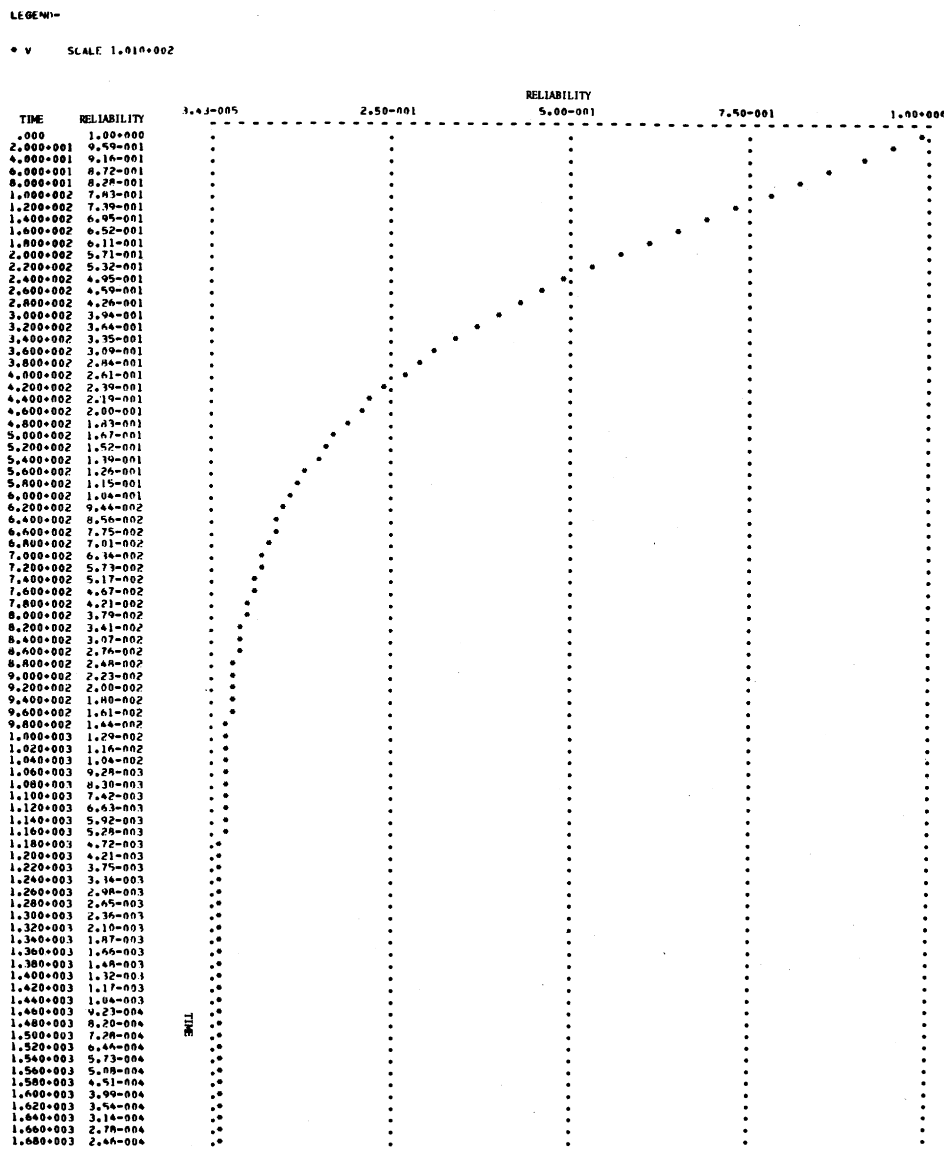
In this paper we present a method to obtain the quantitative evaluation of the availability of a distributed transaction system (DTS).

We introduce some models for the DTS subsystems. The level for the DTS description, which is used, aims at showing how the proposed model works. The actual application of the method to evaluate the availability of a real DTS requires for the application of a "divide and conquer" philosophy. In fact, a bottom-up usage of it allows to evaluate availability for smaller subassemblies of the system and to use these results in the evaluation of larger parts.

Some issues are left open in this work, namely, the evaluation of transition rates in the case of nondirect state dependence, and the impact of the workload and availability on the performance indexes of the system. These problems will be the object of further research in such a way as to allow the application of the proposed model also in wider environments.

Appendix I
Basic Algorithm for Building the Structure Vector

```
for each $q \in Q$
begin Structure Graph Logical Expression SGLE
  begin andchain
    $N_s$ := starting node
    {form an AND chain for each data item}
    for each data item $d_i$ do
        chain[$j$] := $x_{d_i}$ .AND. $x_{s_i}$ .AND. $x_{N_k}$
    if $d_i \in D^k$ and $N_k \neq N_s$ then
        chain[$j$] := chain[$j$] .AND. $x_{lsk}$
  end andchain
```



```
LEGEND-

* V      SCALE 1.010+002

   TIME       RELIABILITY
    .000       1.00+000
 2.000+001     9.59-001
 4.000+001     9.16-001
 6.000+001     8.72-001
 8.000+001     8.28-001
 1.000+002     7.83-001
 1.200+002     7.39-001
 1.400+002     6.95-001
 1.600+002     6.52-001
 1.800+002     6.11-001
 2.000+002     5.71-001
 2.200+002     5.32-001
 2.400+002     4.95-001
 2.600+002     4.59-001
 2.800+002     4.26-001
 3.000+002     3.94-001
 3.200+002     3.64-001
 3.400+002     3.35-001
 3.600+002     3.09-001
 3.800+002     2.84-001
 4.000+002     2.61-001
 4.200+002     2.39-001
 4.400+002     2.19-001
 4.600+002     2.00-001
 4.800+002     1.83-001
 5.000+002     1.67-001
 5.200+002     1.52-001
 5.400+002     1.39-001
 5.600+002     1.26-001
 5.800+002     1.15-001
 6.000+002     1.04-001
 6.200+002     9.44-002
 6.400+002     8.56-002
 6.600+002     7.75-002
 6.800+002     7.01-002
 7.000+002     6.34-002
 7.200+002     5.73-002
 7.400+002     5.17-002
 7.600+002     4.67-002
 7.800+002     4.21-002
 8.000+002     3.79-002
 8.200+002     3.41-002
 8.400+002     3.07-002
 8.600+002     2.76-002
 8.800+002     2.48-002
 9.000+002     2.23-002
 9.200+002     2.00-002
 9.400+002     1.80-002
 9.600+002     1.61-002
 9.800+002     1.44-002
 1.000+003     1.29-002
 1.020+003     1.16-002
 1.040+003     1.04-002
 1.060+003     9.28-003
 1.080+003     8.30-003
 1.100+003     7.42-003
 1.120+003     6.63-003
 1.140+003     5.92-003
 1.160+003     5.28-003
 1.180+003     4.72-003
 1.200+003     4.21-003
 1.220+003     3.75-003
 1.240+003     3.34-003
 1.260+003     2.98-003
 1.280+003     2.65-003
 1.300+003     2.36-003
 1.320+003     2.10-003
 1.340+003     1.87-003
 1.360+003     1.66-003
 1.380+003     1.48-003
 1.400+003     1.32-003
 1.420+003     1.17-003
 1.440+003     1.04-003
 1.460+003     9.23-004
 1.480+003     8.20-004
 1.500+003     7.28-004
 1.520+003     6.46-004
 1.540+003     5.73-004
 1.560+003     5.08-004
 1.580+003     4.51-004
 1.600+003     3.99-004
 1.620+003     3.54-004
 1.640+003     3.14-004
 1.660+003     2.78-004
 1.680+003     2.46-004
```

Fig. 3. Reliability computation.

```
{relations are either partitioned or replicated}
  begin partrel
    {add an AND chain for each partitioned relation}
    SGLE:=TRUE
    for each relation partitioned in $n$ parts do
      for $i$=1,$n$ do
        SGLE:=SGLE .AND. chain[$ji$]
  end partrel
  begin duplrel
    {form an OR chain for each replicated relation}
    for each relation replicated $m$ times do
      if $k$ = 1 then {1-resiliency}
        SGLE:=SGLE .AND. (chain[$j$1] .OR. chain[$j$2] .OR. · · · .OR.
              .OR. chain[$jm$])
      else {with $k$ = $K$ form a chain of $\binom{m}{k}$ chains of $K$ elements}
        SGLE:=SGLE .AND. (chain[$j$1] .AND. · · · .AND. chain[$jK$]) .OR.
              .OR. (chain [ ] .AND.      .AND. chain[ ])
  end duplrel
    SGLE:=SGLE .AND. $x_{N_s}$
end SGLE
```

Appendix II
Numerical Examples

Example 1–Availability Computation: Fig. 2 shows the availability computed for a distributed database with four nodes, with three files, two of which are duplicated, where transition rates are state-independent.

The following parameters are used: for nodes (files): $\lambda = 1 * 10^{-3}$ $\mu = 2.78 * 10^{-2}$ for links: $\lambda = 0.33 * 10^{-3}$ $\mu = 1.6 * 10^{-1}$.

Example 2–Reliability Computation: Fig. 3 shows the reliability (DTS without maintenance) computed for a distributed database with three relations, partitioned in five files, three of which are duplicated, when transition rates are state-dependent. The following parameters are used:

for files:

$$\lambda = 1 * 10^{-3} \qquad \mu = 0$$

$$\gamma_{12} = \gamma_{21} = \gamma_{34} = \gamma_{43} = \gamma_{67} = \gamma_{76} = 2$$

$$\gamma_{53} = \gamma_{54} = \gamma_{86} = \gamma_{87} = 0.5.$$

In both examples all elements are used in the transaction. The parameters λ and μ are constants for elements of the same type.

References

[1] P. A. Alsberg and J. D. Day, "A principle for resilient sharing of distributed resources," in *Proc. 2nd Int. Conf. Software Eng.*, Dec. 1976, pp. 562-570.

[2] M. W. Hammer and D. W. Shipman, "Reliability mechanism for SDD-1, a system for distributed databases," *ACM Trans. Database Syst.*, vol. 5, pp. 431-466, Dec. 1980.

[3] H. Garcia-Molina, "Reliability issues for fully replicated distributed databases," *Computer*, pp. 34-42, Sept. 1982.

[4] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *Proc. ACM Nat. Conf.*, Aug. 1969, pp. 295-305.

[5] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Comput.*, vol. C-27, pp. 540-547, June 1978.

[6] G. Martella, B. Pernici, and F. A. Schreiber, "Distributed data base reliability analysis and evaluation," in *Proc. 2nd Symp. Rel. in Distribut. Software Database Syst.*, Pittsburgh, PA, July 1982, pp. 94-102.

[7] G. Martella, B. Ronchetti, and F. A. Schreiber, "On evaluating availability in distributed database systems," in *Performance Evaluation*, vol. 1, pp. 201-211, Nov. 1981; also in *Proc. 5th Berkeley Workshop on Distributed Data Management Comput. Networks*, Feb. 1981.

[8] I. W. Draffan and F. Poole, *Distributed Data Bases*. New York: Cambridge University Press, 1980.

[9] D. R. Ries and M. Stonebraker, "Locking granularity revisited," *ACM Trans. Database Syst.*, vol. 4, pp. 210-227, June 1979.

[10] K. B. Irani and H. L. Lin, "Queuing network models for concurrent transaction processing in a database system," in *Proc. ACM SIGMOD Conf.*, 1979, pp. 134-142.

[11] D. Potier and P. Leblanc, "Analysis of locking policies in database management systems," *Commun. ACM*, vol. 23, no. 10, pp. 584-593.

[12] L. A. Boone, H. L. Leibergot, and R. M. Sedmark, "Availability, reliability and maintainability aspects of the Sperry Univac 1100/60," in *Proc. 10th Int. Symp. on Fault Tolerant Comput.*, Kyoto, Japan, 1980, pp. 3-8.

[13] X. Castillo and D. P. Siewiorek, "A performance-reliability model for computing systems," in *Proc. 10th Int. Symp. Fault Tolerant Comput.*, Kyoto, Japan, 1980, pp. 187-192.

[14] R. K. Iyer and D. J. Rossetti, "A statistical load dependency model for CPU errors at SLAC," in *Proc. 12th Int. Symp. Fault Tolerant Comput.*, Santa Monica, CA, 1982.

[15] X. Castillo and D. P. Siewiorek, "A workload dependent software reliability prediction model," in *Proc. 12th Int. Symp. Fault Tolerant Comput.*, Santa Monica, CA, 1982.

[16] E. Hansler, G. K. McAuliffe, and R. S. Wilkov, "Exact calculation of computer network reliability," *Networks*, vol. 4, pp. 95-112, 1974.

[17] O. Frank and W. Gaul, "On reliability in stochastic graphs," *Networks*, vol. 12, pp. 119-126, 1982.

[18] E. Zemel, "Polynomial algorithms for estimating network reliability," *Networks*, vol. 12, pp. 439-452, 1982.

[19] J. Spragins, "Reliability problems in data communication systems," in *Proc. 5th Data Commun. Symp., Snowbird*, Sept. 1977, pp. 3.9-3.13.

[20] F. A. Schreiber, "State dependency issues in evaluating distributed database availability," *Comput. Networks*, vol. 8, Sept. 1984.

[21] I. A. Papazoglow and E. P. Gyftopulos, "Markov process for reliability analysis of large systems," *IEEE Trans. Rel.*, vol. R-26, Aug. 1977.

[22] J. A. Buzacott, "Markov approach to finding failure times of repairable systems," *IEEE Trans. Rel.*, vol. R-19, pp. 128-134, Nov. 1970.

[23] M. L. Shooma, *Probabilistic Reliability: An engineering approach*. New York: McGraw-Hill, 1970.

[24] V. Amoia and M. Santomauro, "Computer oriented reliability analysis of large electrical systems," in *Proc. SSCT 77*, Praga, 1977.

[25] V. Amoia, G. De Micheli, and M. Santomauro, "Computer oriented formulation of transition rate matrices via Kronecker algebra," *IEEE Trans. Rel.*, vol. R-30, pp. 123-132, June 1981.

[26] J. Brewer, "Kronecker products and matrix calculus in system theory," *IEEE Trans. Circuits Syst.*, vol. CAS-25, Sept. 1978.

[27] P. Gubian, "Relan: un programma per il calcolo della affidabilita' dei sistemi," M.S. thesis, Politecnico di Milano, Milan, Italy, July 1980.

[28] R. E. Barlow and F. Proschan, *Mathematical Theory of Reliability*. New York: Wiley, 1965.

[29] F. A. Schreiber, "A framework for research in performance-availability of automated information systems," in *Proc. 4th Symp. Rel. Distributed Software and Database Syst.*, Silver Spring, MD, Oct. 1984.

[30] E. Gelenbe and D. Derochette, "Performance of rollback recovery systems under intermittent failures," *Commun. ACM*, vol. 21, pp. 493-499, June 1978.

[31] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Comput. Surveys*, vol. 13, June 1981.

[32] G. Martella, B. Pernici, and F. A. Schreiber, "A reliability model for distributed database systems," C.N.R.-P.F.I. ed., DATANET Res., yellow series, no. 12, Sept. 1982.

Optimal Load Balancing in a Multiple Processor System with Many Job Classes

LIONEL M. NI AND KAI HWANG

***Abstract*–A loosely coupled multiprocessor system contains multiple processors which have their own local memories. To balance the load among multiple processors is of fundamental importance in enhancing the performance of such a multiple processor system. Probabilistic load balancing in a heterogeneous multiple processor system with many job classes is considered in this study. The load balancing scheme is formulated as a nonlinear programming problem with linear constraints. An optimal probabilistic load balancing algorithm is proposed to solve this nonlinear programming problem. The proposed load balancing method is proven globally optimum in the sense that it results in a minimum overall average job response time on a probabilistic basis.**

Manuscript received November 27, 1982; revised December 27, 1984. This work was supported in part by the National Science Foundation under Grant ECS-80-16580.

L. M. Ni is with the Department of Computer Science, Michigan State University, East Lansing, MI 48824.

K. Hwang is with the Computer Research Institute and the Department of Electrical Engineering–Systems, Unviersity of Southern California, Los Angeles, CA 90089.