# Availability analysis of a certain class of distributed computer systems under the influence of hardware and software faults

G.D. Hassapis

*Indexing terms: Fault location, Mathematical techniques*

**Abstract:** There are modelling techniques which may be used to assess the availability of either the software or the hardware of distributed computer systems. However, from the practical point of view, it would be more useful to assess the availability when the system is subjected to the combined effect of hardware and software faults either during its normal operating time or repair time. Such an availability modelling technique, for the class of distributed computer systems which are used for process control, is presented in this paper. To demonstrate the use of this technique the time function of the availability of a typical real-life distributed process control system is evaluated.

## 1 Introduction

The use of distributed computer systems is proliferating in application areas where knowledge of reliability related performance indices is of paramount importance. Such major application areas are process control, space defence and transaction processing. The various mathematical models, which have been proposed by various researchers for the evaluation of these reliability related performance indices of single software packages [1–12] or computer hardware [13–19], may also be used to evaluate the same performance indices of distributed software and hardware. A more accurate evaluation may be made by models proposed especially for distributed programs [20, 21] or distributed system hardware [22–24].

All these modelling techniques, however, do not consider the use of failure time distribution functions of any general form, and they do not take into consideration the combined effect of hardware and software faults. Of course, developing mathematical expressions for the reliability and availability analysis of any distributed computer system in which failure time probability distribution functions have any general form and the combined effect of hardware and software faults is taken into consideration, is extremely difficult. It is proved, however, in this work that such analytical expressions can be developed with relative ease for the availability evaluation of the distributed computer systems, which are used for regulatory process control [28].

Paper 6895C (C2, C3), first received 17th March 1988 and in revised form 23rd March 1989

The author is with the Department of Electrical Engineering, Aristotelian University, P.O. Box 19599, Thessaloniki 540 06, Greece

This work is mainly a generalisation of that of Sumita and Masuda [12] who have developed analytical mathematical expressions for evaluating the time function of the availability of a single software package, the operation or repair of which is interrupted by hardware failures. This theory has been made more appropriate for the type of software used in the distributed process control systems and has been extended by incorporating the state of the computer hardware at time $t$ explicitly. Also, a heuristic algorithm has been developed for the numerical evaluation of the derived analytical expressions. To demonstrate the use of the proposed analysis technique, the availability of a distributed computer system of the type used in the process industry for regulatory control is studied.

## 2 Distributed process control systems

Available distributed computer systems, which are classified as distributed process control systems, follow a hierarchical and highly modular hardware architecture. They are also loaded with a library of software routines which are able to perform well defined control, monitoring and plant management functions. Usually, such a system consists of computer modules which are organised in four levels, according to the dedicated function they have been programmed to perform. These are the input/output level, the data acquisition and control level, the network level and the operator's console/supervisory computer level. At the input/output level, each computer module of this level, usually corresponding to a printed circuit card, execute continuously a software routine which controls the input and output of a number of analog and digital signals and transfers their values through a local data bus to a specific module of the data acquisition and control level. Each one of the microcomputer based modules of the data acquisition and control level performs under the control of a routine filtering, smoothing, conversion to engineering units of the received input signals, control calculations for a group of process variables and database updating of the modules of the operator's console/supervisory computer level through the modules of the network level. The modules of the operator's console/supervisory computer level provide a human interface to the operator and allow the execution of higher level applications, such as performance optimisation, energy management, sequencing and scheduling. A basic characteristic of the system is that a data file is not transferred upon a request of the receiving routine but its transfer is controlled entirely by the routine which generates it.

**Table 1: Software routines of the system and the computer modules required to run each routine.**

| No | Program $i \in [1, 6]$ | Hardware modules $CP-i$ | $I-i$ | $I/O-i$ | $NI-1$ | $NP-1$ | $SC-1$ |
|---|---|---|---|---|---|---|---|
| 1 | configurable control function | * | | | | | |
| 2 | $I/O$ handling | | | * | | | |
| 3 | data acquisition | | * | | | | |
| 4 | network control | | | | | * | |
| 5 | data base manager | | | | * | | |
| 6 | operating system | | | | | | * |

The design features of the distributed process control system described above now allow us to view the system operation at each timing instant as the parallel running of a number of software routines on an equal number of computers. Obviously, if a computer failure occurs, this will not influence the logic of a routine which runs on another computer. The same applies for routines which have internal errors. A routine failure influences only the data that it may send to other routines, but it cannot corrupt the other routines. In this sense, the operation of a computer or a routine may be considered to be independent of the operation of another computer or routine, respectively.

Of course, considering a fully parallel operation of routines is a simplification of the actual system operation. In fact, there are instances at which the parallel operation is interrupted, i.e. when a routine wishes to store a data file in the memory of another computer. The overall time, however, of the non-parallel running of the routines may be ignored, as it is negligible when it is compared to the time the routines run in parallel. This simplification and some additional assumptions, explained in the following and concerning the initial state of the system and the statistical characteristics of the hardware and software components of the system, assist us in deriving a mathematically manageable availability model.

## 3 Availability analysis

The availability of any system is defined as the probability of finding the system in operation at any given time. Depending on the way the various components of the system interact during the system operation, this probability may be expressed as a specific function of the probabilities of finding the individual components of the system in operation. Therefore, the availability of a distributed process control system can be evaluated by considering the probabilities of finding its computer modules and its software routines in operation at a given time. These probabilities can be evaluated as follows.

First, let us assume that the time that elapses from the moment a computer module has failed until the beginning of its repair is negligible. Then the state of each computer module in time may be expressed by the stochastic process:

$$J_i(t) = \begin{cases} 0, & \text{if the } i\text{th computer module is under repair at time } t. \\ 1, & \text{if the } i\text{th computer module is functioning at time } t. \end{cases}$$

The state, however, of a software routine cannot be expressed by a single stochastic process since the routine may be in an operational state and have internal errors. These errors may cause a routine failure only under specific operating conditions, which may not appear during the routine lifetime. Further, during the repair of the routine after a failure, new errors may be introduced, although the error which caused the failure has been rectified. Practice, however, indicates that one should expect the reduction of the initial number of the routine internal errors when software repair has repeatedly taken place. Taking into consideration these routine peculiarities we may express mathematically the state of a software routine by the vector of stochastic processes $[I_i(t), M_i(t), N_i(t)]$, where:

$$I_i(t) = \begin{cases} 0, & \text{if the } i\text{th software routine is under repair at time } t. \\ 1, & \text{if the } i\text{th software routine is functioning at tine } t. \end{cases}$$

$$M_i(t) = \begin{cases} \text{the number of failures occurred} \\ \text{in } [0, t) \text{ in the } i\text{th software routine.} \end{cases}$$

$$N_i(t) = \begin{cases} \text{the number of errors in the } i\text{th} \\ \text{software routine at time } t. \end{cases}$$

According to the system functional behaviour previously described, the operation of any software routine does not depend on whether the other routines are operational. Also, the operation of any computer module is independent of the operation of the other modules. Then, the probability of being the entire system in an operational state, according to the multiplication theorem of the probability theory, can be expressed by:

$$P[I_1(t) = 1 \cdots I_L(t) = 1, M_1(t) = m_1 \cdots M_L(t) = m_L, N_1(t) = n_1 \cdots N_L(t) = n_L, J_1(t) = 1 \cdots J_L(t) = 1] = \prod_{i=1}^{L} P[I_i(t) = 1, M_i(t) = m_i, N_i(t) = n_i] P[J_i(t) = 1] \quad (1)$$

There will be, however, more than one operational states for each routine, each state corresponding to a different value of the vector of stochastic processes of the routine. Therefore, according to the addition theorem of the probability theory, the probability of being the system in any one of its possible operational states will be:

$$A(t) = \sum_{n_1}^{\infty} \sum_{n_2}^{\infty} \cdots \sum_{n_L}^{\infty} \sum_{m_1}^{\infty} \sum_{m_2}^{\infty} \cdots \sum_{m_L}^{\infty} \prod_{i=1}^{L} p1_{i, m_i, n_i}(t) \Pi 1_i(t) \quad (2)$$

where

$$pj_{i, m_i, n_i}(t) = P[I_i(t) = j, M_i(t) = m_i, N_i(t) = n_i] \quad (3)$$

$$\Pi j_i(t) = P[J_i(t) = j] \quad (4)$$

and $j = 0, 1$, $i = 1, 2, \ldots, L$, $L$ is the number of the software routines.

This probability, however, according to the definition given above, is the time function of the system availability. To evaluate this function it is necessary first to evaluate the individual $p1_{i, m_i, n_i}(t)$ and $\Pi j_i(t)$ probabilities.

## 4 Evaluation of state probabilities

Deriving at analytic expression for $p1_{i, m_i, n_i}(t)$ directly is rather difficult. Instead, we may derive such analytic expressions in an easier way if we consider that these probabilities are the final probabilities of the states that the random vector process, which describes the behaviour of a software routine, is capable of assuming. Then, these probabilities can be expressed in terms of state

entry rates for which analytic expressions are already available.

If we assume that:

(*a*) $f0_{i,m_i,n_i}(t)$ is the entry rate of a software routine from the state $(1, m_i - 1, n_i)$ to the state $(0, m_i, n_i)$

(*b*) $f1_{i,m_i,n_i}(t)$ is the entry rate of the same routine from the state $(0, m_i, n_i)$ to the state $(1, m_i, n_i)$.

(*c*) the uptime and downtime of the routine may have any general probability distribution, denoted by $a_{i,n_i}(t)$ and $r_{i,n_i}(t)$, $i = 1, 2, \ldots, L$ respectively.

(*d*) The uptime distribution of any computer is assumed to be $e(t) = \lambda_0 \exp(-\lambda_0 t)$ while down time may have any general distribution $h_i(t)$, $i = 1, 2, \ldots, L$.

(*e*) The probability that there are $n_i$ errors remaining in a software routine after a repair, given that there were $k$ errors just before the beginning of the repair is $p_{kn_i}$

(*f*) The maximum number of latent errors that may exist in a software routine is limited to a number $K$, then it is proved [12] that:

$$f0_{i,m_i,n_i}(t) = \int_0^t f1_{i,m_i-1,n_i}(x-y) aeff_{i,n_i}(x)\,dy \tag{5}$$

$$f1_{i,m_i,n_i}(t) = \sum_{k=1}^{K} p_{kn_i} \int_0^t f0_{i,m_i,k}(x-y) \times reff_{i,k}(y)\,dy \tag{6}$$

where $aeff_{i,n_i}(t)$ and $reff_{i,n_i}(t)$ are the effective software up and down time distributions respectively, computed by the relations:

$$aeff_{i,n_i}(t) = \sum_{g=0} \int_0^t \left(\exp(-\lambda_0 x)\frac{(\lambda_0 x)^g}{g!}\right) \times a_{i,n_i}(x) h_i^{(g)}(x-y)\,dy \tag{7}$$

$$reff_{i,n_i}(t) = \sum_{g=0} \int_0^t \left(\exp(-\lambda_0 x)\frac{(\lambda_0 x)^g}{g!}\right) r_{i,n_i}(x) \times h_i^{(g)}(x-y)\,dy \tag{8}$$

where $h_i^{(g)}(t)$ is the $g$-fold convolution of $h_i(t)$ with itself $h_i^{(0)}(t) = \delta(t)$ and $g \in [0, \ ]$ now represents the number of computer module failures.

Now, the state probabilities can be easily related to the entry rates by subtracting from the probabilistic flow that went into a state, the flow that went out of this state. That is:

$$p1_{i,m_i,n_i}(t) = \int_0^t f1_{i,m_i,n_i}(x)\,dx - \int_0^t dx \int_0^t f1_{i,m_i,n_i}(x-y) \times aeff_{i,n_i}(y)\,dy \tag{9}$$

$$p0_{i,m_i,n_i}(t) = \int_0^t f0_{i,m_i,n_i}(x)\,dx - \int_0^t dx \int_0^t f0_{i,m_i,n_i}(x-y) \times reff_{i,n_i}(y)\,dy \tag{10}$$

Similar expressions can be derived for the probabilities $\Pi 1_i(t)$ if we assume that:

(*a*) $\phi 1_i(t)$ is the entry rate of the $i$th computer module from its operational state to the failed state, and

(*b*) $\phi 0_i(t)$ is the entry rate of the same module from the failed state to the operational state.

For a computer module to enter a failed state at time $t$ it should have entered the operational state at time $t - x$, the up-time of which expires at $x$, $0 < x < t$. This implies that:

$$\phi 0_i(t) = \int_0^t \phi 1_i(x-y) e(y)\,dy \tag{11}$$

Similarly, for a computer module to enter the operational state it should have entered a failed state at $t - x$, the down time of which expires at $x$, $0 < x < t$. Therefore, it will be:

$$\phi 1_i(t) = \int_0^t \phi 0_i(x) h_i(x-y)\,dy \tag{12}$$

Then, the state probabilities of the module will be:

$$\Pi 0_i(t) = \int_0^t \phi 0_i(x)\,dx - \int_0^t dx \int_0^t \phi 1_i(x-y) h_i(y)\,dy \tag{13}$$

$$\Pi 1_i(t) = \int_0^t \phi 1_i(x)\,dx - \int_0^t dx \int_0^t \phi 1_i(x-y) e(y)\,dy \tag{14}$$

Having found analytical expressions for the state probabilities it is now possible to express the time function of the system availability in terms of these probabilities and evaluate it in the way demonstrated in the next section.

## 5 Computation procedure

As one can note the evaluation of the various state probabilities involves the computation of multiple convolutions. An algorithmic framework for the evaluation of multiple convolutions is provided by the Laguerre transform [25]–[27]. For example, if we assume that $a(x)$ is a square integrable rapidly decreasing function, then this function can be expressed in the series form:

$$a(x) = \sum_{n=0}^{\infty} a_n^+ L_n(x) \tag{15}$$

where

$$a_n^+ = \sum_{m=0}^{n} \hat{a}_m \tag{16}$$

$L_n(x)$ is the Laguerre function defined as:

$$L_n(x) = \exp(-x/2)(\exp(x)/n! + (d/dx)^n x^n \exp(-x)) \tag{17}$$

and $\hat{a}_m$, $m = 0, 1, \cdots n$, are the so-called Laguerre sharp coefficients the values of which can be found by equating the terms of the same power of $u$ in the relationship:

$$\sum_{n=0}^{\infty} \hat{a}_n u^n = A(u) \tag{18}$$

$A(u)$ is the function that is derived if in the Laplace transform of $a(x)$, say $A(s)$, we let $s = (1+u)/2(1-u)$.

For the convolution $c(x) = \int_0^t a(x-y) b(y)\,dy$ holds:

$$\hat{c}_n = \sum_{j=0}^{n} \hat{a}_{n-j} \hat{b}_j \tag{19}$$

where $\hat{c}_n$ and $\hat{b}_n$ are respectively the sharp coefficients of $c(x)$ and $b(x)$.

Also, it is proved that the sharp coefficients of the integral $W(x) = \int_x^{\infty} a(y)\,dy$ are related to the sharp coefficients of $a(x)$. That is:

$$\hat{W}_n = -2\hat{a}_n + 4\sum_{j=0}^{\infty} (-1)^j \hat{a}_{n+j} \tag{20}$$

It has been shown [12] that the Laguerre sharp coefficients of $f1_{i,m_i+1,n_i}(t)$ can be obtained from the recursive formula:

$$\hat{f1}_{i,m_i+1,n_i;j} = \sum_{l=0}^{j} \sum_{k=0}^{K} \hat{f1}_{i,m_i,k;l} \hat{\hat{z}}_{l,k;j-l} \tag{21}$$

where $\hat{z}_{i,k,ni:j-l}$ is the $j-l$ sharp coefficient of the Laplace transform function:

$$Z_{i,k}(s) = Aeff_{i,k}(s)^{*}Reff_{i,k}(s) \tag{22}$$

To apply this recursive formula initial values must be given to the coefficients $\hat{f}_{i,0,ni:0}$ and $\hat{f}_{i,0,ni:l}$. Such values can be found by considering the fact that at system delivery all the routines are in an operational state and the latent errors of each routine can cause a routine failure at a later time when the system will operate under actual plant operating conditions. Therefore, it will be:

$$\hat{f}_{i,0,ni:0} = 1 \quad \text{and} \quad \hat{f}_{i,0,ni:l} = 0 \tag{23}$$

Once the coefficients of $f1_{i,mi,ni}(t)$ are found the corresponding coefficients of $f0_{i,mi,ni}(t)$ can be computed by applying (19). Similarly, by considering (19), (11) and (12) one easily derives that:

$$\hat{\phi}0_{i:j} = \frac{\sum_{q=0}^{j-1} \hat{\phi}1_{i:q}\hat{e}_{j-1-q} + \hat{e}_0 \sum_{q=0}^{j-1} \hat{\phi}0_{i:q}\hat{h}_{i:j-1-q}}{\hat{1} - \hat{e}_0 h_{i:0}} \tag{24}$$

$$\hat{\phi}1_{i:j} = \frac{\sum_{q=0}^{j-1} \hat{\phi}0_{i:q}\hat{h}_{i:j-1-q} + \hat{h}_0 \sum_{q=0}^{j-1} \hat{\phi}1_{i:q}\hat{e}_{j-1-q}}{1 - \hat{e}_0 \hat{h}_{i:0}} \tag{25}$$

It is also:

$$\int_0^t a(y)\,dy = W(0) - W(t) + a(0) \tag{26}$$

Then, by using (19), (20) and (26) and setting

$$p1_{i,0,K}(0) = 1,\ p0_{i,mi,ni}(0) = 0$$

for $m_i \in [1, K]$ and

$$\Pi 1_{i,mi,ni}(0) = 1, \quad \Pi 0_i(0) = 0,$$

for the reasons explained previously, the integral functions (9), (10), (13) and (14) can be evaluated for all the computer modules and software routines, if a finite number of terms is defined for the infinite series truncation.

A heuristic algorithm may be used to truncate the various state probabilities in such a way that the overall truncation error of the availability is bounded by a specified error tolerance $\varepsilon$. This algorithm is developed on the basis of the following observations.

$$E < 1 - \sum_{n_1=0}^{K} \cdots \sum_{n_L=0}^{K} \sum_{m_1=0}^{K} \cdots \sum_{m_L=0}^{K} \prod_{i=1}^{L} p1a_{i,mi,ni} \times (t)^{*}\Pi 1a_i(t) < \varepsilon \tag{27}$$

where $E$ is the overall truncation error and $pla_{i,mi,ni}(t)$ and $\pi la_i(t)$ are the truncated time series of the respective state probabilities of the software routines and computer modules. This relationship will be true if:

$$\prod_{l=1}^{L} p1a_{i,mi,ni}(t)^{*}\Pi 1a_i(t) > \frac{1-\varepsilon}{(K+1)^{2L}-1} \tag{28}$$

Then, the heuristic algorithm can perform a stepwise increase of an initial number of truncation terms up to the point of finding the number of truncation terms which make each truncated state probability to satisfy the relationship (28).

## 6 An application example

Let us consider the distributed computer control system shown in Fig. 1. In Table 1, the computer modules required to run each software routine are shown. Without any loss of generality but in order to provide reasonable graphical representation of the estimated state probabilities, we assume that all the computer modules and hardware modules have the same statistical characteristics. They are provided in Table 2. Programming the already discussed algorithmic procedure on an IBM 4381/13 mainframe computer in Fortran 77, it was made possible to evaluate the time function of the availability of the considered distributed computer system with a $+0.001$ error tolerance. Graphical representation of this function and the corresponding state probabilities are given in Fig. 2 and 3, respectively.

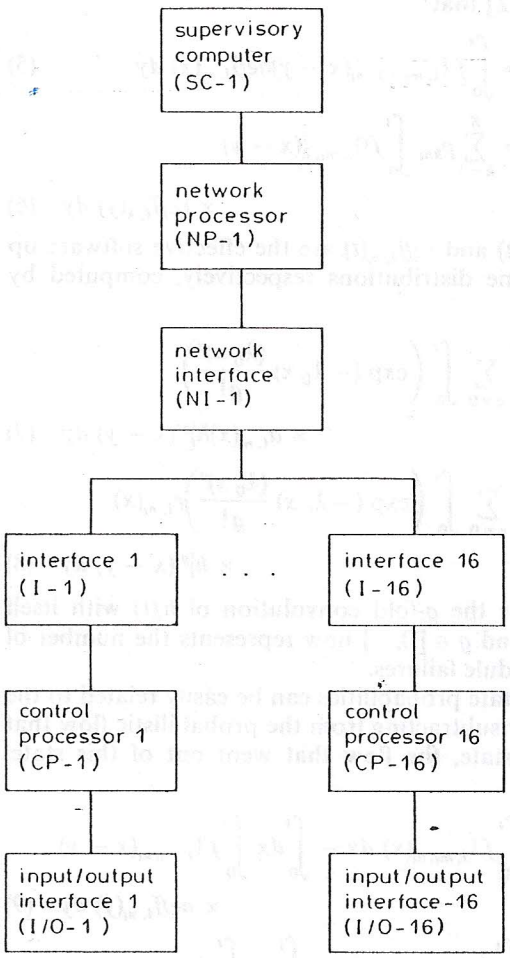


**Fig. 1** *Block diagram of the distributed computer control system.*

**Table 2: Statistical features of the distributed processing system ($i = 1, 2, \cdots, 6$)**

$$h_i(t) = 0.5[\exp(-t) + 2\exp(-2t)]$$

$$r_{i,n_i}(t) = \frac{t}{(4-n_i)!}\exp(-t)\ \ n_i = 1, 2,\ r_{i,0}(t) = 0$$

$$a_{i,n_i}(t) = \frac{\gamma_{n_i}\beta_{n_i}}{\gamma_{n_i} - \beta_{n_i}}(\exp(-\gamma_{n_i}t) - \exp(-\beta_{n_i}t)),\ n_i = 1, 2$$

$$a_{i,0}(t) = 0$$

$$\gamma_{n_i} = 1.5,\ \beta_{n_i} = 1.8, 1.7 \text{ for } n_i = 1, 2$$

$$\lambda_0 = 0.1$$

$$P = (P_{kn_i}) = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.600 & 0.350 & 0.050 \\ 0.450 & 0.520 & 0.030 \end{bmatrix}$$

Each curve in Fig. 2 corresponds to a different number of errors in each software routine. Specifically the number 1 curve was obtained by considering that each one of the six software routines may have up to 2 errors. The number 2 curve considers that there are only two routines with up to 2 errors whereas the others are free of errors. The last curve refers to a system having only one routine with two errors. These different curves demonstrate the expected strong dependence of the system availability on the number of routines with errors.

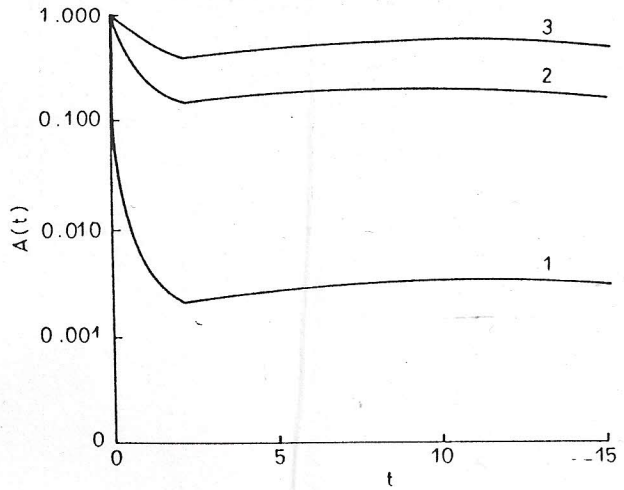


**Fig. 2** *System availability curves.*

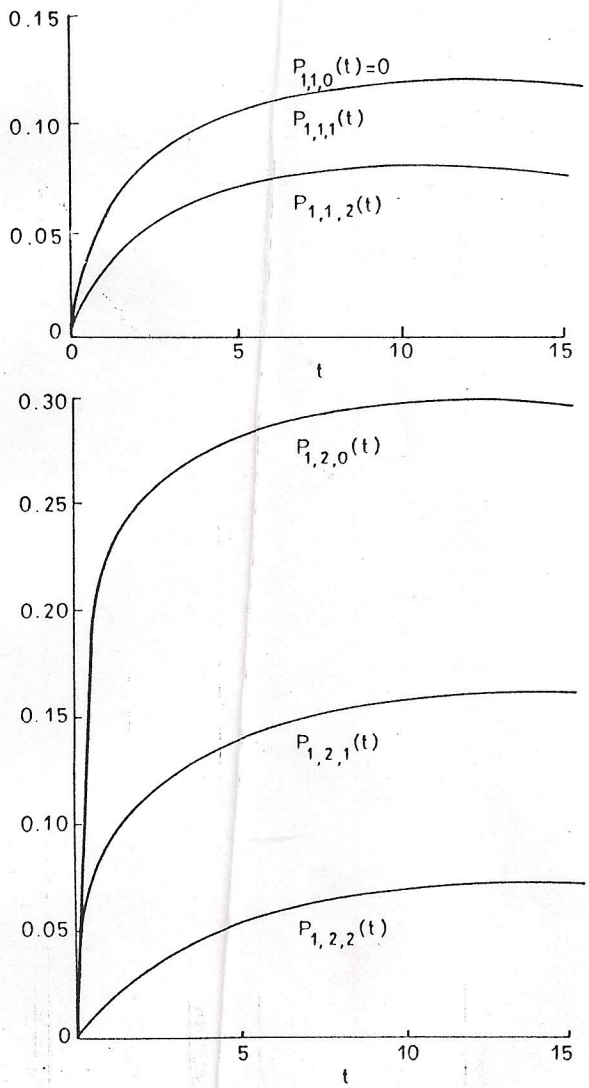


**Fig. 3** *State probabilities of a software package.*

## 7 Conclusions

It has been proved that the availability of the distributed process control systems, which constitute a class of the general category of distributed computer systems, can be evaluated within prespecified error bounds by expressing analytically the operational state probabilities of each software and hardware module of the system, and computing them by a heuristic algorithm. The algorithmic computations utilise the Laguerre transform to evaluate the multiple convolutions of the analytical expressions. The initially considered number of errors in each software module influences the system state space which becomes enormous for large numbers. This, combined with a very small truncation error of the availability estimation results to large memory requirements. The memory capacity requirement may be reduced by trading the number of errors per software package with the accuracy of the availability estimation, which defines the number of terms in their series approximation.

## 8 References


1 DUANE, J.T.: 'Learning curve approach to reliability monitoring', *IEEE Trans.*, **AES-2**, pp. 379–410
2 JELINSKI, Z., and MORANDA, P.B.: 'Software reliability research', *in* 'Statistical computer performance evaluation', ed. W. Freiberger, Academic Press, New York, 1972, pp. 465–484
3 LITTLEWOOD, B., and VERALL, J.L.: 'A Bayesian reliability growth model for computer software'. J. Royal Stat. Soc. C, **22**, 1973, pp. 332–346
4 CROW, L.: 'Reliability analysis for complex, repairable systems' in 'Reliability and Biometry', ed. F. Proscham and R.J. Serfling, Philadelphia, **PA: SIAM**, 1974, pp. 379–410
5 MUSA, J.D.: 'A theory of software reliability and its application', *IEEE Trans.*, **SE-1**, 1975, pp. 312–327
6 GOEL, A.L., and OKUMOTO, K.: 'Time-dependent error detection rate model for software reliability and other performance measures', *IEEE Trans.*, **R-28**, 1979, pp. 206–211
7 LITTLEWOOD, B.: 'Stochastic reliability growth: a model for fault removal in computer programs and hardware designs', *IEEE Trans.*, **R-30**, 1981, pp. 313–320
8 RAMAMOORTHY, C.V., and BASTANI, F.: 'Software reliability model-status and perspective', *IEEE Trans.*, **SE-8**, 1982, pp. 354–371
9 SHANTIKUMAR, J.C.: 'On a'software reliability model: a review', *Microelectron. and Rel.*, **23**, 1984, pp. 903–943
10 MILLER, D.R.: 'Exponential order statistic models of software reliability growth', *IEEE Trans.*, **SE-12**, 1986, pp. 12–24
11 SCHOLZ, F.W.: 'Software reliability modeling and analysis', *IEEE Trans.*, **SE-12**, 1986, pp. 25–31
12 SUMITA, U., and MASUDA, Y.: 'Analysis of software availability/reliability under the influence of hardware failures', *IEEE Trans.*, **SE-12**, 1986, pp. 32–41
13 BEAUDRY, M.D.: 'Performance-related reliability measures for computing systems', *IEEE Trans.*, **C-27**, 1978, pp. 540–547
14 MEYER, J.F., FURCHTGOTT, D.G., and WU, L.T.: 'Performability evaluation of the SIFT computer', *IEEE Trans.*, **C-29**, 1980, pp. 501–509
15 MEYER, J.F.: 'On evaluating the performability of degradable computing systems', *IEEE Trans.*, **C-29**, 1980, pp. 720–731
16 HUSLENDE, R.: 'A combined evaluation of performance and reliablity for degradable systems', *Perf. Eval. Rev.*, **10**, 1981, pp. 157–164
17 DONATIELLO, L., and IYER, B.R.: 'Analysis of a composite performance reliability measure for fault-tolerant systems', *Journal of ACM*, **34**, 1987, pp. 179–199
18 GOYAL, A., and TANTAWI, A.N.: 'Evaluation of performability for degradable computer systems', *IEEE Trans.*, **C-36**, 1987
19 DE SUZA, E., SILVA, E., and GAIL, H.R.: 'Calculating cumulative operational time distributions of repairable computer systems', *IEEE Trans.*, **C-35**, 1986, pp. 322–332
20 LITTLEWOOD, B.: 'Software reliability model for modular program structures', *IEEE Trans.*, **R-28**, 1979, pp. 241–246
21 PRASANNA KUMAR, V.K., HARISI, S., and RAGHAVENDRA, C.S.: 'Distributed program reliability analysis', *IEEE Trans.*, **SE-12**, 1986, pp. 42–50
22 SIEWIOREK, D.P., KINI, V., JOOBANI, R., and BELLIS, H.: 'A

case study of C.mmp, Cm*, and C.vmp: Part II-Predicting and calibrating reliability of multiprocessor systems'. Proceedings of IEEE, **66**, 1978, pp. 1200–1220
23 GARCIA-MOLINA, H., and KENT, J.: 'Evaluating response time in a faulty distributed computing system', *IEEE Trans.*, **C-34**, 1985, pp. 101–109
24 GOEL, A.L., and SOENJOTO, J.: 'Models for hardware-software system operational-performance evaluation', *IEEE Trans.*, **R-30**, 1981, pp. 232–239
25 KEILSON, J., and NUNN, W.R.: 'Laguerre transformation as a tool for numerical solution for integral equation of convolution type', *Appl. Math. Comput.*, **5**, 1979, pp. 313–359
26 KEILSON, J., NUNN, W.R., and SUMITA, U.: 'The bilateral Laguerre transform', *Appl. Math. Comput.*, **8**, 1981, pp. 137–174
27 SUMITA, U.: 'The matrix Laguerre transform', *Appl. Math. Comput.*, **15**, 1984, pp. 1–28
28 SCHOEFFLER, J.D.: 'Distributed computer systems for industrial process control', *IEEE Comput.*, **14**, 1984, p. 11