Dependability Modeling and Performability Evaluation of Software Executed on Correlated Inputs

Peter Popov
Bulgarian Academy of Sciences, Institute of Computer and Communication Systems
Acad. G. Bontchev street, bl.2, 1113 Sofia, Bulgaria
phone: (+359 2) 71 90 97, fax: (+359 2) 72 39 05
e-mail: ptp@sun.iccs.acad.bg

1. Introduction

The main scope of this paper is dependability modeling and performability evaluation of software executed on a succession of non independently drawn points of the input space (iterative software). "Bursts of failures" are encountered in such software. They result from running the software along trajectories (successions of points closely located to each other in the multi-dimentional space) and their crossing the existing "failure regions" [2]. In this context, the failure rate alone does not exhaustively characterise the software in respect to dependability, nor suffice to carry out a quantitative analysis of dependability. A model for adequate modeling the failure occurrence process for such software is presented that addresses both aspects of dependability: i) the interfailure time and ii) the sojourn time in failure. The assumption of independent selection of inputs, that is inappropriate for the iterative software (process control applications and similar), is substituted for an independence in selecting trajectories in the model.

The model allows for the well-developed *theory of renewal processes* to be used as a mathematical framework in dependability modeling and performability evaluation.

In section 2 a summary of the known models that address the dependability of iterative software is given and the new model is presented. The differences of the model from models reported earlier by others are emphasised. The model is argued to depend on measurable parameters and ways of how to carry out measurements are briefly discussed. In section 3 a sketch of performability evaluation of a software system depending on measurable parameters is presented. Finally, in section 4 conclusions and some suggestions for future research are outlined.

2. Reliability Modeling of Software Execuited on Correlated Inputs

2.1. Previous Results

One of the first attempts to model the bursts of failures in software operation has been presented in a paper by Csenki [2]. In his model two kinds of failure events are considered - *spontaneous* failure that is the first failure occurring after a succession of successful executions and *certain* failures immediately following the spontaneous failure. The number of certain failures depends upon the size of the encountered failure region and is modelled as a random variable x not exceeding a given value σ. Formally:

$$P(x=i)=p_i;\ i=0,1,\ldots,\sigma;\ \sum_{i=0}^{\sigma} p_i = 1. \qquad (1)$$

In the simplest case (of discrete time) the system behavior is represented by a discrete Markov chain in which all failures - spontaneous and certain - are assigned particular states. One of these, S, represents the spontaneous failure and other σ states, F_1, F_2, ... F_σ, represent the certain failures. One additional state, O, represents the successful execution of the software.[1] The transition probabilities between states are assumed known that describe the model:
p_{OS} - the failure rate;

[1]In [2] the model is more complicated since recovery blocks are modelled. Failure regions are modeled consistently with our quotation here.

p_{SF_i} - conditional probability of a series of *i* certain failures given spontaneous failure has happened, i=0,1,..., σ;

$p_{F_1F_2}, p_{F_2F_3},..., p_{F_{\sigma-1}F_\sigma} = 1$.

The same approach has been used in [5] and has been slightly extended in [1].

The model may be criticised for:

- implicitly assuming the interfailure time being exponentially (geometrically) distributed, that is a special case of interfailure time [6],
- depending on unknown distribution of the sojourn time in failure that poses limitations of practicability of the analysis.

2.2. Our Model

2.2.1. Assumptions and Informal Description

The model presented below is based on three essential assumptions:

- the duration of a single software execution (on a particular input data) is assumed *fixed*, Δt. This is a very typical case in process control applications. In order to ensure that the time constraints are fulfilled, intervals between software executions guarantee that the execution will be completed on every possible input data within Δt.
- the process of control consists of *processing* a *set of tasks* having, in general, *different* duration of processing (measured in number of consecutive executions).
- a set of predefined events in the software environment trigger the set of implemented tasks - each event is handled by a task being invoked. The order of events, and hence the order of invoking the tasks' (responding to them), is assumed random. Moreover, we assume that the events are *independent* of each other having in mind the usual statistical meaning of independence. This assumption seems plausible in applications as those reported by von Mayerhauser et al. in [3] and similar.

Assumptions' justification is omitted here because of space limitations and could be found in [7].

Assume that the distributions of the time needed for processing every task (measured in number of software executions) is known. Also assume that the probability of selecting at random each task is known. Then the (marginal) distribution of the processing time of a *randomly selected task* could be defined. A series of tasks' being processed now can be treated as a multiple processing of the same *randomly selected task*. Thus the series of tasks could be represented as a renewal process formed by processing randomly selected tasks. We would like to stress again that the independence between tasks is essential for representing the series of tasks as a renewal process. If the independence is not acceptable, then the renewal process is not a correct abstraction representing the behavior of the system.

The *randomly selected task* is executed on a series of inputs of the input space. Thinking of this the task is now *mapped onto the input space*. Thus in the abstract model of software behavior the independently selected for processing entities (tasks) could be substituted for the successions of inputs on which the tasks will be executed (spending a fixed time Δt on each input) while being under processing. The succession of points on which the software is executed while a particular task is being processed forms a *trajectory* (associated with the task) trough the input space.

2.2.2. Probabilistic Description of the Model

Let S_T be the set of all possible trajectories in a given application and *Tr(•)* be the probabilistic measure defined on S_T , i.e. *Tr(i)* be the probability that a randomly selected trajectory $Lt \in S_T$ happens to be *i*. *Tr(•)* defines the operational profile of the iterative software. Let lt_t *(•)* be the probability distribution function of trajectories' length, measured in number of software executions each lasting Δt.

$$lt_t(t = i) = \sum_{j \in S_T} lt_t(t = i | Lt = j).Tr(j) \qquad (2)$$

Note that $lt_t(t = i | Lt = j)$ will be a Dirac function at point *i* corresponding to the actual length of the trajectory.

For each program π, S_T may be represented as a union of two non-overlapping subsets S_1 and S_2. The set S_1 encompasses all trajectories none of them crossing a failure region (non-faulty trajectories) while the set S_2 - all trajectories crossing a failure region (faulty trajectories). Formally, $S_T = S_1 \cup S_2$ and $S_1 \cap S_2 = \varnothing$.

Let us introduce the following notations:

$$1-\theta = \sum_{j \in S_1} Tr(j) \text{ and } \theta = \sum_{j \in S_2} Tr(j). \qquad (3)$$

where θ is the probability that a randomly selected trajectory (in accordance with *Tr(•)*) crosses a failure region, i.e. happens to belong to S_2, while 1-θ is the probability to select at random a trajectory from S_1 (non-faulty trajectory).

We could express the conditional distributions of trajectories' length of both subsets S_1 and S_2:

$$lt_t(t|S_1) = \sum_{j \in S_1} lt_t(t|j).Tr(j)$$
$$lt_t(t|S_2) = \sum_{j \in S_2} lt_t(t|j).Tr(j) \qquad (4)$$

Let $b(t|\pi,j)$[2] be the probability distribution of the *sojourn time in failure* of program π along trajectory j. Then:

$$b(t|\pi) = \sum_{j\in S_T} b(t|\pi, j)Tr(j)\,. \qquad (5)$$

Conditional distributions of the sojourn time in failure for S_1 and S_2, respectively, could be expressed as:

$$\begin{aligned} b(t|S_1) &= \sum_{j\in S_1} b(t|\pi, j)Tr(j) \\ b(t|S_2) &= \sum_{j\in S_2} b(t|\pi, j)Tr(j) \end{aligned} \qquad (6)$$

Thus:

$$b(t|\pi) = (1-\theta).b(t|S_1) + \theta.b(t|S_2)\,. \qquad (7)$$

In the last formulae and in what follows the program index π is omitted for sake of brevity. Since by definition: $b(t|S_1) = \begin{cases} 1, & t = 0, \\ 0, & elsewhere \end{cases}$ and the mean value of this is :

$$\mu|S_1 = \int_0^\infty t.b(t|S_1)dt = 0\,,$$

then the mean value of the marginal sojourn time in failure will be:

$$\begin{aligned} \mu(\pi) &= \int_0^\infty t.b(t|\pi)dt = (1-\theta).\int_0^\infty t.b(t|S_1)dt + \\ &\theta.\int_0^\infty t.b(t|S_2)dt = 0 + \theta.\mu|S_2 = \theta.\mu|S_2 \end{aligned}\,. \qquad (8)$$

where $\mu|S_2 = \int_0^\infty t.b(t|S_2)dt$ is the expected value of the sojourn time in failure for the set of faulty trajectories.

In the model two sources of uncertainty (variability) are incorporated: the length of a randomly selected trajectory is subject to variability and so is the number of failures along it. Thus the model can degenerate to special cases when either of these sources of variability is removed or even both of them. Software operating with independent selection of inputs is a special case of a software in which the second source of variability is not present. On each input the software either fails or succeeds, i.e. at most one failure could be observed (no variability of the sojourn time in failure). The first source of variability in such a software is existent [4]. If the execution time is constant and software is executed on independently drawn inputs, then both sources of variability disappear.

2.2.3. Interfailure Time

The system operation can be expressed as a series of *cycles*, each encompassing a *random number* of non-faulty trajectories (picked from S_1), and exactly *one* faulty trajectory (from S_2). The number of non-faulty trajectories, N, in a cycle is geometrically distributed and therefore $P(N=n) = \theta.(1-\theta)^n$. Let NF be the random variable denoting the *total duration* of the non-faulty trajectories within a cycle. Denote its *pdf*[3] as $w(t)$. Let CY be the random variable denoting the cycle duration with *pdf* $w_{cycle}(t)$, and finally, let $L_1^*(s)$, $L_2^*(s)$, $W^*(s)$ and $W_{cycle}^*(s)$ be the Laplace transforms of $lt_t(t|S_1)$, $lt_t(t|S_2)$, $w(t)$ and $w_{cycle}(t)$, respectively.

Obviously:

$$\begin{aligned} W^*(s) &= \theta\sum_{i=1}^{\infty}\left(L_1^*(s).(1-\theta)\right)^i \\ W_{cycle}^*(s) &= \theta\sum_{i=1}^{\infty}\left(L_1^*(s).(1-\theta)\right)^i.L_2^*(s) \end{aligned} \qquad (9)$$

NF and *CY* set up bounds for the interfailure time, that in practical cases should be tight. While $lt_t(t|S_1)$, in principle could be measured with arbitrary accuracy, and hence determining $w(t)$ is not a problem, $lt_t(t|S_2)$ is practically impossible to measure and therefore $w_{cycle}(t)$ can hardly be estimated. In order to overcome this impediment in estimating $w_{cycle}(t)$ there are at least two ways to go:

- *CY* is substituted for *NF* (use $w(t)$ instead of $w_{cycle}(t)$) and the duration of the faulty trajectory in a cycle is completely ignored. Since, in practice N could be expected to be large, the error will be negligible. Thus, $w(t)$ serves for interfailure time estimation (lower bound).
- make additional assumptions about $lt_t(t|S_2)$, for example assume that $lt_t(t|S_2)$ and $lt_t(t|S_1)$ are identical, that may be very difficult to justify.

In both cases, using measurable $lt_t(t|S_1)$ (its estimate $\hat{lt}_t(t|S_1)$) and some hypothesis about θ, the interfailure time distribution can be expressed. At this point we stress that software with (conditionally, given θ) exponential

[2] For a program π that along j crosses failure regions only once and stays there for k executions or twice and stays there for k_1 and k_2 , respectively:

$$b(t|\pi, j) = \begin{cases} 1 & for\ \ t = k, \\ 0 & elsewhere. \end{cases} \qquad b(t|\pi, j) = \begin{cases} 1/2 & for\ \ t = k_1, \\ 1/2 & for\ \ t = k_2, \\ 0 & elsewhere. \end{cases}$$

[3] Strictly, by definition all distributions related to trajectories' length are discrete. Since the set S_T is expected to be immensely large, however, the continuous approximation of all distributions seems a reasonable transformation. Therefore the pdfs' and their Laplace transforms could be used without serious consequences.

distribution of the interfailure time is *a special case of a software*, although a very important special case. In our model this restrictive assumption is removed and, from this point of view, the model may be considered as a generalisation of the Csenki model. In the general case, whether the geometric (exponential) distribution is an appropriate approximation of the actual interfailure time depends on the relationship between θ and the first two moments of $lt_t|S_1$. Details are available in [6].

2.2.4 Sojourn Time in Failure - Relation with the Probability of Events.

In practice, after observing a failure, an attempt will be made to locate the fault having caused the failure and the fault will eventually be removed. Thus, the usual situation after the testing is completed will be to have no observation of failures caused by the remaining software faults. We may only conjecture that certain number of faults remains still unrevealed in the software and expect due to them some residual failure rate in operation. Since the faults are not known it is difficult to express some educated guess about the characteristics of those faults, in particular something about the size of their failure regions.

In the model of independent trajectories, by definition:

$b(t = k)$ = P(select at random a k-faulty trajectory),

for every k of interest.

Suppose we are able to collect failure data that along with the time between failures also include a description of how long (number of consecutive executions) software stays in failure for each observed spontaneous failure. If so, we will be able to predict, through reliability growth models, the probability of the event “selecting at random a k-faulty trajectory” for every k of interest. The task of collecting such data, obviously, is not trivial. The aspects of how this could be achieved is beyond the scope of this paper and is an area for future research. Here we assume that failure data allow for observed intervals between consecutive k-faulty trajectories to be recovered. These constitute the observed history of k-faulty trajectories. Thus reliability growth models could be applied to each such history and predictions of the (probability distribution of the) time to next k-faulty trajectory could be obtained. This together with the estimation of the mean trajectory length will give prediction of the probability to select at random a k-faulty trajectory and, hence $b(t = k)$ will be determined. Thus, when the testing is completed we will be able to express an *educated guess* about the distribution of the sojourn time in failure.

2.2.5 The Role of Measurement in Determining the Model Parameters

Variability of trajectories’ length could influence the interfailure time distribution and should be assessed through measurements [6]. More sophisticated methods and tools for failure data collection (than those currently being used) are needed for accurately predicting the distribution of the sojourn time in failure.

3. Performability evaluation of Software Executed on Correlated Environment

3.1. Assumptions

The basic assumptions used in performability evaluation are as follows:

- the model of software executed on independently selected trajectories, described above, is applicable.
- the system is operating for a time T, referred to as a *mission time* which is short compared with the mean value of the interfailure time. In other words, during the mission time it is very unlikely to observe a failure.
- failures are classified as *benign* when the sojourn time in failure is shorter than a predefined value n_c and *catastrophic* otherwise [1]. The model of software failures used in the analysis is a subset of the model presented in [1]. The assumption’s justification is omitted for sake of brevity and could be found in [7].
- performability is assessed as the expected value of a reward function M(t) at the end of the mission E[M(T)]. The reward is defined as follows [4]:

1. M(0) = 0, the reward is nullified at the beginning of the mission;

2. Each software execution (lasting Δt) contributes to the reward function value in accordance with the rules:

$$\begin{cases} +1, \text{ if the execution is correct,} \\ +0, \text{ in case of a benign failure, and} \\ M(T) = 0, \text{ in case of catastrophic failure (the mission is invalidated).} \end{cases}$$

We assume the distribution of the sojourn time in failure $b(x|\pi)$ known (determined through reliability growth predictions) and denote this for short as $b(x)$.

3.2. Calculations

Details of performability related calculations are omitted because of space limitations. These are available in [7]. The frame of the approach being used is only sketched below:

1. The expected value of the reward function in case of a successful mission, R(T), is calculated. In this case the sojourn time in failure *b(x)* is substituted for the sojourn time in (benign) failure, $b(x|x<n_c)$, that is:

$$b(x|x<n_c)=\frac{b(x,x<n_c)}{1-P(x\geq n_c)}, \qquad (10)$$

where $P(x\geq n_c)=\int_{n_c}^{\infty}b(x)dx$.

2. The probability of a successful mission, P_{ncf} is determined as the probability of the time between catastrophic failures being longer than the mission time (determined from the *predicted distribution* of the time between catastrophic failures).
3. Since the reward is nullified in case of a catastrophic failure, the unconditional expected value of the accrued reward during a mission (performability) is calculated as:

$$E[M(T)] = (1 - P_{ncf}) . 0 + P_{ncf} . R(T). \qquad (11)$$

The distribution of trajectories' length together with the predicted sojourn time in failure and distribution of the time to next catastrophic failure suffice for both R(T) and P_{ncf} to be calculated. All calculations necessary in the analysis are dependent on measurable parameters only.

4. Conclusions

In this paper some issues of modeling the reliability of software systems executed on correlated inputs are addressed.

An approach to modeling the software behavior as a series of tasks independently selected for processing, each requiring a bounded time for processing, is presented. When mapped onto the input space this abstraction yields a series of independently selected trajectories each comprising closely located points of the input space. The model allows two aspects of software dependability to be adressed. First, the effect of the varying trajectory length upon the interfailure time to be accounted for (the interfailure time is not necessarily exponentially distributed and could be determined through measurements). Second, the distribution of the sojourn time in failure is objectively determined using the well-developed theory of software reliability growth modeling. These two features, as it seems to me, partially solve the problems of using the abstract models for evaluating dependability and performability of real software systems executed on correlated inputs.

The work could be enlarged in different ways. First, developing tools for data collection purposed at estimating the probability distribution of trajectories' length and predicting the sojourn time in failure. Second, theoretical generalization of the model in order to tackle the behavior of fault-tolerant software executed on correlated inputs is needed.

References

[1] Bondavali, A., Chiaradonna, S., Di Giandomenico, F., and Strigini, L., "Dependability Models for Iterative Software Considering Correlation among Successive Inputs", 1994

[2] Csenki, A., "Recovery Block Reliability Analysis with Failure Clustering", Dependable Computing and Fault-Tolerant Systems, A. Avizienis, J.-C. Laprie eds., Vol. 4, Springer-Verlag, pp. 75-103.

[3] von Mayerhauser, A., Walls, J., and Mraz, R., "Testing Applications Using Domain Based Testing and Sleuth", ISSRE'94.

[3] Tai, A. T-A. Performability Concepts and Modeling Techniques for Real-Time Software, UCLA, 1991, Ph.D. dissertation, 151 p.

[4] Tai, A. T., Meyer, J. F., Avizienis, A., "Performability Enhancement of Fault-Tolerant Software", IEEE Trans. Reliability, Vol. 42, No. 2, pp. 227-237, June 1993.

[5] Tomek, L. A., Muppala, J. K., and Trivedi, K. "Modeling Correlation in Software Recovery Blocks", IEEE Trans. Software Engineering, Vol. 19, No. 11, November 1993, pp. 1071 - 1086.

[6] Popov, P., "The effect of execution time variability on the software reliability growth modelling", to appear in the proceedings of the Second European Conference of Dependable Computing EDCC-2, October 1996, Taormina, Italy.

[7] Popov, P., "Dependability Modelling and Performability Evaluation of Software Executed on Correlated Inputs", Technical Report, ICCS-BAS, 1996, http://w3.iccs.acad.bg.