1-1-2007

Reliability of fault-tolerant systems with parallel task processing

Gregory Levitin
Israel Electric Corporation Ltd

Min Xie
National University of Singapore

Tieling Zhang
National University of Singapore, tieling@uow.edu.au

Reliability of fault-tolerant systems with parallel task processing

Abstract

The paper considers performance and reliability of fault-tolerant software running on a hardware system that consists of multiple processing units. The software consists of functionally equivalent but independently developed versions that start execution simultaneously. The computational complexity and reliability of different versions are different. The system completes the task execution when the outputs of a pre-specified number of versions coincide. The processing units are characterized by different availability and processing speed. It is assumed that they are able to share the computational burden perfectly and that execution of each version can be fully parallelized. The algorithm based on the universal generating function technique is used for determining the distribution of system task execution time. This algorithm allows analysts to evaluate complex hardware-software reliability and performance indices such as expected task execution time and probability that the task is completed within a given time. Illustrative examples are also presented.

Keywords

task, processing, parallel, systems, reliability, tolerant, fault

Disciplines

Engineering | Science and Technology Studies

Publication Details

Levitin, G., Xie, M. & Zhang, T. (2007). Reliability of fault-tolerant systems with parallel task processing. European Journal of Operational Research, 177 (1), 420-430.

Reliability of Fault-Tolerant Systems with Parallel Task Processing

Gregory Levitin
Reliability Department, Planning, Development and Technology Division,
Israel Electric Corporation Ltd., P.O. Box 10, Haifa, 31000 Israel

Min Xie, Tieling Zhang
Dept of Industrial and Systems Engineering
National University of Singapore, Singapore 117576

Abstract

The paper considers performance and reliability of fault-tolerant software running on a hardware system that consists of multiple processing units. The software consists of functionally equivalent but independently developed versions that start execution simultaneously. The computational complexity and reliability of different versions are different. The system completes the task execution when the outputs of a pre-specified number of versions coincide. The processing units are characterized by different availability and processing speed. It is assumed that they are able to share the computational burden perfectly and that execution of each version can be fully parallelized.

The algorithm based on the universal generating function technique is used for determining the distribution of system task execution time. This algorithm allows analysts to evaluate complex hardware-software reliability and performance indices such as expected task execution time and probability that the task is completed within a given time. Illustrative examples are presented.

Keywords: Performance; Fault-tolerant software; Multiprocessor system; Universal generating function.

1. Introduction

Modern architecture of computer systems presumes software execution by a system of highly interconnected hardware units that have ability to share the computational task in an effective way [1-7]. The examples of such hardware configuration are processors in multiprocessor systems, computational resources in Grid computing systems [8,9] and the recently emerged configurable system-on-chip architecture consisting of a set of heterogeneous processing resources and a reconfigurable processing cell array [10]. In such complex hardware-software systems, both hardware and software components are failure-prone. Carrying out reliability analysis of such systems is important although it is not straightforward.

For such systems, hardware unavailability is typically caused by failures of the electronic equipment or by external impacts. Effective self-diagnostics and maintenance activity usually allows the unavailable units to be restored or repaired in a short time.

Software failures are caused by errors made in various phases of the development. When the software reliability is of critical importance, special software design and development techniques are used to achieve fault tolerance. Two of the best-known fault-tolerant software design methods are N-version programming (NVP) and recovery block scheme (RBS) [11]. Both methods are based on the redundancy of software modules (functionally equivalent but independently developed) and the assumption that coincident failures of modules are rare. Many research works have been devoted to the study of fault-tolerant system's reliability [4-8, 12-20]. The fault tolerance usually requires additional resources and results in performance penalties (particularly with regard to computation time), which constitutes a tradeoff between software performance and reliability. This effect has been studied in [21-24].

The combination of fault-tolerant software methodology with effective multiprocessor hardware architecture allows system designers to meet both reliability and performance requirements.

NVP was proposed by Chen and Avizienis [25]. This approach presumes the execution of n functionally equivalent software modules (called versions) is able to receive the same input and send their outputs to a voter that is aimed at determining the system output. The voter produces an output if at least m out of n outputs agree (it is assumed that the probability that m wrong outputs agree is negligibly small). Otherwise, the system fails. Usually majority voting is used in which n is odd and $m = (n+1)/2$.

RBS was proposed by Randell [26]. In this approach, after execution of each version, its output is tested by an acceptance test block (ATB). If the ATB accepts the version output, the process is terminated and the version output becomes the output of the entire system. If all n versions can not produce the accepted output, the system fails. It was shown in [24] that when the acceptance test time is included into the execution time of each version, the RBS performance model becomes identical to the performance model of the NVP with $m = 1$.

Since the performance of fault-tolerant programs depends on hardware processing speed (which in its turn depends on availability of computational resources), the impact of hardware availability should be taken into account when the system performance and availability are evaluated. This paper presents an algorithm for finding the reliability and performance measures for arbitrary fault-tolerant hardware-software systems based on multiple processing units with perfect task sharing. The presented algorithm is straightforward and fast. It does not take into consideration imperfect software task parallelization and existence of common cause failures in both hardware and software. However it can be useful as a theoretical framework for further development of more sophisticated models. It can also be applied to fast evaluation of the upper bound of system

performance, which is important when different system designs are compared or when system configuration optimization problems are solved in which approximate estimates of system performance should be obtained for large number of different solutions.

Unlike works [5-7], this paper does not deal with software version scheduling. It suggests a methodology for studying the effect of characteristics of both software versions and hardware units on reliability and performance of the entire fault-tolerant system. In the presented model it is assumed that all the software versions are executed by the hardware units in parallel. However, the same methodology can be used for analysis of systems with arbitrary schedule of versions' execution (different schedules change version termination times but do not affect the probabilities that the task is completed after execution of a given number of versions). The main advantage of the suggested method is its ability to take into account limited availability and diversity of hardware components, while in [5-7] it was assumed that all the hardware components (processors) are identical and fully reliable.

Acronyms & Notations

PU	processing unit
pmf	probability mass function
u-function	moment generating function
NVP	N-version programming
RBS	recovery block scheme
$1(x)$	unity function: 1(TRUE) = 1, 1(FALSE) = 0
n	total number of software versions
m	number of versions that should produce correct results
p_k	probability that m correct outputs are obtained after termination of version k
N	number of processing units

x_i	processing speed of PU i
a_i	availability of PU i (usually measured as probability that any given task assigned to the PU can be successfully executed or as a fraction of the PU is available)
S	random cumulative processing speed of hardware system
s_j	j-th realization of S
J	total number of different realizations of S
q_j	probability $\Pr(S = s_j)$
Θ	random time of task execution by the system
θ^*	maximum allowed time of task execution by the system
$R(\theta^*)$	probability that system successfully terminates its task in time less than θ^*
c_i	computational complexity of version i
r_i	reliability of version i
g_k	computational complexity of stage k of task execution
H	random computational complexity of software task (amount of computations performed until m correct outputs are obtained)
h_k	amount of computations until termination of version k
T_k	random termination time of stage k
t_{kj}	termination time of stage k when $S = s_j$
Q_{kj}	probability that the task terminates after stage k when $S = s_j$
W	conditional expected system execution time
$D(z), V(z), U(z)$	u-functions representing distributions of discrete random variables
MTBF	mean time between failures
MTTR	mean time to restoration

2. Model formulation and preliminary results

2.1. System structure and assumptions

The system structure can be described as follows. A hardware system consists of N different statistically independent processing units (PUs). Each PU i is characterized by its availability a_i and performance (processing speed) in operational state x_i. It should be noted that the availability of PUs depends on the availability of inter-processor communication channels. The probability of communication failures should be taken into account when evaluating the availability of PUs. Usually PUs are much more reliable than software modules, however, unreliability of the communication channels can make the software reliability and hardware availability to be of the same magnitude. In the simplest case (for example, in Grid networks with star architecture [9]) communication link and processing unit can be modeled by two independent elements connected in series. In more complex cases the unavailability of communication channels can cause common cause failures. The algorithm presented in this paper considers systems without common cause hardware failures; however the suggested universal generating function technique can easily be adapted for incorporating this type of failures [31].

It is assumed that the hardware system is able to distribute the computational task among the available processors in the most effective way (perfect task sharing) such that any task can be fully parallelized. This assumption is realistic when each task (software version) can be divided into subtasks executed in parallel (for example, in Grid networks the resource management system can divide computational task and send them for parallel execution to available processors). Even when the perfect parallelization of the computational task is impossible, the presented model can be used as a tool for fast evaluation of the upper bound of system performance.

Under the assumption of perfect parallelization the cumulative system processing speed (measured in number of basic computer operations executed per time unit) is equal to the sum of processing speeds of available PUs. Since the combination of available processors is random, the cumulative system processing speed S is a random value that can have J different realizations.

Each software version j is characterized by its computational complexity c_j (required number of basic computer operations) and reliability r_j (probability of producing correct output). The versions start their execution simultaneously. The task execution is divided among the PUs in such a way that the versions proceed with equal speed.

It is assumed that any task execution time is much smaller than MTBF and MTTR of the PUs and, therefore, the probability that the PUs can change their state during software task execution is negligibly small.

Let us order the versions according to their computational complexity: $c_j \le c_{j+1}$ ($1 \le j \le n - 1$). In the first stage of the computational task execution (from the beginning till the termination of version 1) the total computational complexity of task that should be performed is $g_1 = nc_1$ (all n versions are executed in parallel until c_1 operations are performed in each of them). In the second stage (after termination of the first version and till termination of the second one) the total computational complexity is $g_2 = (n - 1)(c_2 - c_1)$. Indeed, in order to finish the second version, the system should perform the remaining $(c_2 - c_1)$ operations of this version (performing in parallel the same amount of operations for each not completed version). The number of versions running simultaneously during the second stage is $(n - 1)$. It can be seen that the total computational complexity of stage k is

$$g_k = (n - k + 1)(c_k - c_{k-1}) \quad \text{for } 1 \le k \le n, \tag{1}$$

where $c_0 = 0$ by definition given in Fig. 1.

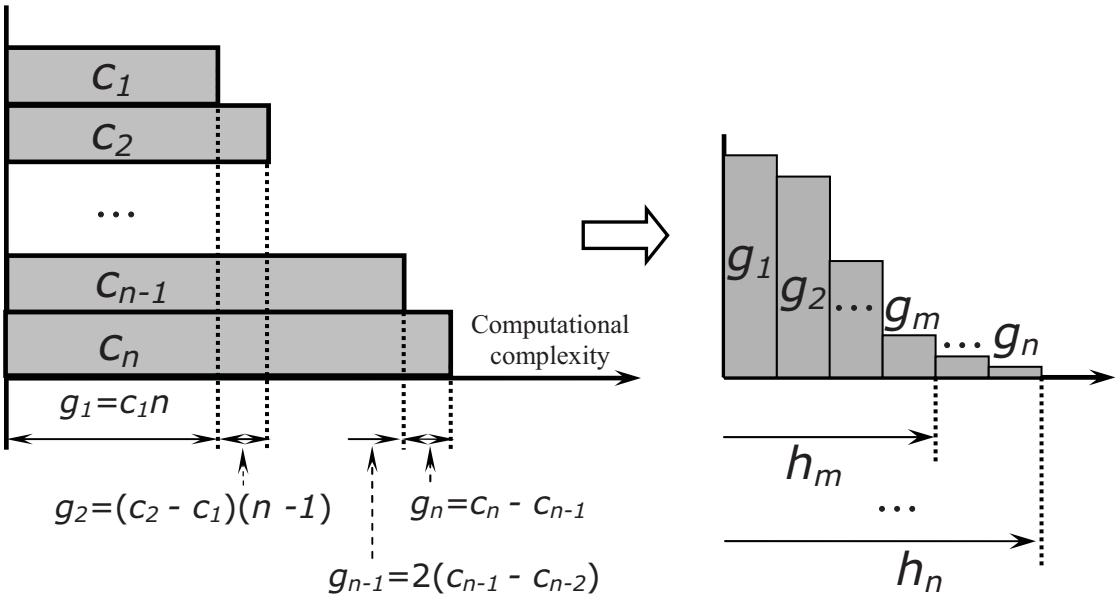

Fig. 1 Computational complexity of computation task

The amount of computations till termination of stage k can be obtained as

$$h_k = \sum_{i=1}^{k} g_i \,. \tag{2}$$

It can be seen that assuming h_0=0 one can obtain h_k using the following recursive expression:

$$h_k = h_{k-1} + g_k = h_{k-1} + (n-k+1)(c_k - c_{k-1}) \tag{3}$$

for $1 \le k \le n$.

The number of versions that should terminate in order to produce m correct outputs can vary from m (m first versions produce correct outputs) to n (nth version produces mth correct output). Therefore, the total amount of calculations that should be performed till task termination H is a random variable depending on outputs of individual versions. H can take values of h_k for $m \le k \le n$.

For the given cumulative system processing speed S, the time of stage k termination is

$$T_k = \frac{h_k}{S} \,. \tag{4}$$

For each realization s_j of the random system processing speed S, one can obtain the realization of random termination time of stage k:

$$t_{kj} = \frac{h_k}{s_j}. \tag{5}$$

Having the probability $q_j = \Pr(S = s_j)$ and the probability p_k that the system produces the correct output after termination of stage k: $p_k = \Pr(H = h_k) = \Pr(\Theta = T_k)$, one can obtain the probability that the total time of task execution Θ is equal to t_{kj} :

$$Q_{kj} = \Pr(\Theta = t_{kj}) = \Pr(H = h_k \mid S = s_j) = q_j p_k, \tag{6}$$

which gives the probability mass function of the random task execution time Θ in the form of pairs (t_{kj}, Q_{kj}) for $m \le k \le n$ and $1 \le j \le J$.

2.2. System reliability and expected system execution time

In order to estimate both the system's reliability and its performance, different measures can be used depending on the application. In applications where the execution time of each task is of critical importance, the system reliability $R(\theta^*)$ is defined (according to performability concept [21, 27, 28]) as a probability that the correct output is produced in less time than θ^*. This index can be obtained as

$$R(\theta^*) = \sum_{j=1}^{J} \sum_{k=m}^{n} Q_{kj} \cdot 1(t_{kj} < \theta^*). \tag{7}$$

In applications where the average system productivity (the number of executed tasks) over a fixed mission time is of interest [28], the system reliability is defined as the probability that it produces correct outputs without respect to the total execution time. This index can be referred to as $R(\infty) = \Pr(\Theta < \infty)$ (which is equal to the probability that the random execution time is not greater than its maximal finite realization). The conditional

expected system execution time W (given the system produces correct output) is considered to be a measure of its performance.

This index determines the expected execution time of the system given that the system does not fail. It can be obtained as

$$W = \sum_{j=1}^{J} \sum_{k=m}^{n} Q_{kj} t_{kj} \,/\, R(\infty). \tag{8}$$

In order to calculate the both indices $R(\theta^*)$ and W, one has to obtain the pmf of the random cumulative processing speed of the hardware system in the form (s_j, q_j) for $1 \le j \le J$ and the pmf of the software task complexity in the form (h_k, p_k) for $m \le k \le n$. The following section presents algorithms for determining these distributions and the distribution of the total task execution time Θ.

3. Algorithms for determining the task execution time distribution

The procedure used in this paper for the system survivability evaluation is based on the universal generating function (u-function) technique, which was introduced in [29] and proved to be very effective for the reliability evaluation of different types of multi-state systems [30, 31].

The u-function representing the pmf of a discrete random variable Y is defined as a polynomial

$$u(z) = \sum_{k=1}^{K} \alpha_k z^{y_k}, \tag{9}$$

where the variable Y has K possible values and α_k is the probability that Y is equal to y_k.

To obtain the u-function representing the pmf of a function of two independent random variables $\varphi(Y_i, Y_j)$, composition operators are introduced. These operators determine the u-

function for $\varphi(Y_i, Y_j)$ using simple algebraic operations on the individual u-functions of the variables. All of the composition operators take the form

$$U(z) = u_i(z) \underset{\varphi}{\otimes} u_j(z) = \sum_{k=1}^{K_i} \alpha_{ik} z^{y_{ik}} \underset{\varphi}{\otimes} \sum_{h=1}^{K_j} \alpha_{jh} z^{y_{jh}} = \sum_{k=1}^{K_i} \sum_{h=1}^{K_j} \alpha_{ik} \alpha_{jh} z^{\varphi(y_{ik}, y_{jh})} \quad (10)$$

The polynomial $U(z)$ represents all of the possible mutually exclusive combinations of realizations of the variables by relating the probabilities of each combination to the value of function $\varphi(Y_i, Y_j)$ for this combination.

3.1. Hardware system processing speed distribution

In the case of hardware systems, the u-function $u(z)$ can define performance distributions of the processing units. Each PU i can have performance x_i (with probability a_i) when it is available and performance 0 (with probability $1 - a_i$) when it is not available. Therefore, the u-function of this PU takes the form:

$$u_i(z) = a_i z^{x_i} + (1 - a_i) z^0. \quad (11)$$

The total processing speed of a pair of PUs is equal to the sum of the processing speeds of these PUs. To obtain the u-function representing the performance distribution of a subsystem containing two PUs, i and j, a composition operator with $\varphi(Y_i, Y_j) = Y_i + Y_j$ should be used. In this case, the operator obtains the product of the corresponding polynomials:

$$u_i(z) \underset{+}{\otimes} u_j(z) = [a_i z^{x_i} + (1 - a_i) z^0][a_j z^{x_j} + (1 - a_j) z^0]. \quad (12)$$

For all N PUs, the pmf of their cumulative processing speed can be obtained as

$$U(z) = \prod_{i=1}^{N} u_i(z) = \prod_{i=1}^{N} [a_i z^{x_i} + (1 - a_i) z^0] = \sum_{j=1}^{J} q_j z^{s_j}, \quad (13)$$

where J is the number of different possible realizations s_j of the total system processing speed, which is equal to the number of terms in $U(z)$ obtained after collecting the like terms

(collecting the like terms corresponds to obtaining the overall probability of different combinations of available PUs that produce the same cumulative processing speed).

$U(z)$ can be obtained recursively in $U_e(z)$, where

$$U_1(z) = u_1(z) \text{ and } U_k(z) = U_{k-1}(z)\, u_k(z) \text{ for } k = 2, \ldots, N. \tag{14}$$

3.2 Software task complexity distribution

Let b_j be an indicator of the success of version j such that $b_j = 1$ if the version produces the correct output and $b_j = 0$ if it produces the wrong output. The distribution of b_j can be represented by the u-function

$$v_j(z) = r_j z^1 + (1 - r_j)\, z^0. \tag{15}$$

It can be seen that the product of polynomials,

$$V_k(z) = \prod_{j=1}^{k} v_j(z) = \prod_{j=1}^{k} [r_j z^1 + (1 - r_j) z^0] = \sum_{j=0}^{k} \omega_j z^j \tag{16}$$

represents the distribution of the number of correct outputs after the execution of a group of first k versions. Indeed the resulting polynomial relates the probabilities of combinations of correct and wrong outputs (the product of corresponding probabilities) with the number of correct outputs in these combinations (the sum of success indicators). Note that after collecting the like terms (corresponding to obtaining the overall probability of different combinations with the same number of correct outputs) the coefficient ω_j in $V_k(z)$ is equal to the probability that the group of first k versions produces exactly j correct outputs.

The function $V_k(z)$ can also be obtained by using the recursive expression

$$V_k(z) = V_{k-1}(z) v_k(z) = V_{k-1}(z)[r_k z^1 + (1 - r_k) z^0]. \tag{17}$$

According to its definition, p_k is the probability that the group of first k versions produces m correct outputs given the group of first $k - 1$ versions has produced $m - 1$ correct outputs,

while ω_m in u-function $V_k(z)$ is equal to the unconditional probability that the group of first k versions produces m correct outputs.

In order to let the coefficient ω_m in u-function $V_k(z)$ be equal to p_k, the term with exponent being equal to m should be removed from $V_{k-1}(z)$ before applying Eq. (17) (excluding the combination in which k first versions produce m correct outputs while the *k-th* version fails). The above considerations are based on the following algorithm for determining all of the probabilities p_k ($m \le k \le n$). It has the following three steps:

1. Determine the u-function of each version according to Eq. (15);
2. Define $V_0(z) = 1$;
3. For $k = 1, 2, \ldots, n$,

   3.1. Obtain $V_k(z)$ using Eq. (17) and collecting similar terms;

   3.2. If $k \ge m$, assign $p_k = \omega_m$;

   3.3. Remove term $\omega_m z^m$ from $V_k(z)$ (if such a term exists).

The combination of values of h_k obtained using recursive expression (3) and values of p_k obtained by the presented algorithm for $m \le k \le n$ constitutes the pmf of task complexity H. This pmf can also be represented in the form of a u-function:

$$D(z) = \sum_{k=m}^{n} p_k z^{h_k} . \tag{18}$$

The presented algorithm for determining the probabilities p_k is based on the assumption that failures in different versions of software are statistically independent. This assumption usually oversimplifies the fault-tolerant software model and gives optimistic evaluation of its reliability. In order to incorporate the common cause failures, one can use a more sophisticated algorithm suggested in [32].

3.3 Total task execution time distribution

The obtained probability mass functions of two independent random variables S and H are represented by two u-functions $U(z)$ and $D(z)$, respectively. In order to obtain the pmf of random task execution time $T = H/S$, we can use the following composition operator over $U(z)$ and $D(z)$:

$$D(z) \underset{\varphi}{\otimes} U(z) = \sum_{k=m}^{n} p_k z^{h_k} \underset{\varphi}{\otimes} \sum_{j=1}^{J} q_j z^{s_j} = \sum_{k=m}^{n} \sum_{j=1}^{J} p_k q_j z^{h_k / s_j} = \sum_{k=m}^{n} \sum_{j=1}^{J} Q_{k\,j} z^{t_{kj}}. \quad (19)$$

Since the term with $s_j = 0$ (simultaneous failure of all of the PUs) corresponds to failure in task execution, it can be removed from $U(z)$ before performing the operator (19).

The resulting u-function represents the distribution (t_{kj}, Q_{kj}) for $m \le k \le n$ and $1 \le j \le J$. Allying Eqs. (7) and (8) over this distributions, one can obtain the system reliability and performance indices $R(\theta^*)$, $R(\infty)$ and W.

4. Illustrative Examples

Example 1

Consider a fault-tolerant software with $n = 5$ and $m = 3$ running on hardware system consisting of two PUs. The availability and processing speed (in mega-operations per second) of each unit are presented in Table 1. The reliability and computational complexity (in mega-operations) of software versions are presented in Table 2.

Table 1 Parameters of PUs given in Example 1

No. of PU j	1	2
a_j	0.9	0.8
x_j	4	6

Table 2 Parameters of software versions for Example 1

Version i	1	2	3	4	5
r_i	0.7	0.6	0.8	0.6	0.9
c_i	6	7	10	12	13

In order to determine the system processing speed distribution, define the u-functions of individual PUs according to Eq. (11):

$$u_1(z) = 0.9z^4 + 0.1z^0;\ u_2(z) = 0.8z^6 + 0.2z^0.$$

The u-function representing the distribution of the system cumulative processing speed takes the form:

$$U(z) = u_1(z)\, u_2(z) = (0.9z^4 + 0.1z^0)\,(0.8z^6 + 0.2z^0) = 0.72z^{10} + 0.08z^6 + 0.18z^4 + 0.02z^0.$$

After removing the term corresponding to the total hardware system failure, we have

$$U(z) = 0.72z^{10} + 0.08z^6 + 0.18z^4.$$

Now consider the software system and determine the amount of computations till termination of each stage h_k according to Eq. (3):

$$h_0 = 0,\ h_1 = 0 + 5(6 - 0) = 30,\ h_2 = 30 + 4(7 - 6) = 34,\ h_3 = 34 + 3(10 - 7) = 43,$$

$$h_4 = 43 + 2(12 - 10) = 47,\ h_5 = 47 + 1(13 - 12) = 48.$$

According to the given parameters, define the u-functions of software versions following Eq. (15):

$$v_1(z) = 0.3z^0 + 0.7z^1;\ v_2(z) = 0.4z^0 + 0.6z^1;\ v_3(z) = 0.2z^0 + 0.8z^1;$$

$$v_4(z) = 0.4z^0 + 0.6z^1;\ v_5(z) = 0.1z^0 + 0.9z^1.$$

According to the algorithm presented in Section 3.2, determine the probabilities p_j:

$$V_0(z) = 1;\ V_1(z) = 1 \cdot v_1(z) = 0.3z^0 + 0.7z^1;$$

$$V_2(z) = V_1(z)\, v_2(z) = (0.3z^0 + 0.7z^1)\,(0.4z^0 + 0.6z^1) = 0.12z^0 + 0.46z^1 + 0.42z^2;$$

$$V_3(z) = V_2(z) v_3(z) = (0.12z^0 + 0.46z^1 + 0.42z^2)(0.2z^0 + 0.8z^1) = 0.024z^0 + 0.188z^1 + 0.452z^2 + 0.336z^3.$$

Remove the term $0.336z^3$ from $V_3(z)$ and obtain $p_3 = 0.336$. We have

$$V_4(z) = V_3(z)\, v_4(z) = (0.024z^0 + 0.188z^1 + 0.452z^2)\,(0.4z^0 + 0.6z^1) =$$

$$0.0096z^0 + 0.0896z^1 + 0.2936z^2 + 0.2712z^3.$$

Remove the term $0.2712z^3$ from $V_4(z)$ and obtain $p_4 = 0.2712$. Continuously, we have

$$V_5(z) = V_4(z)\, v_5(z) = (0.0096z^0 + 0.0896z^1 + 0.2936z^2)\,(0.1z^0 + 0.9z^1)$$

$$= 0.00096z^0 + 0.0176z^1 + 0.11z^2 + 0.26424z^3.$$

Finally, we obtain $p_5 = 0.26424$.

Having the values of h_k and p_k, determine the u-function $D(z)$ representing the pmf of H:

$$D(z) = 0.336z^{43} + 0.2712z^{47} + 0.26424z^{48}\,.$$

Obtain the u-function representing the task execution time distribution:

$$D(z) \underset{\varphi}{\otimes} U(z) = (0.336z^{43} + 0.2712z^{47} + 0.26424z^{48}) \underset{\varphi}{\otimes} (0.72z^{10} + 0.08z^6 + 0.18z^4)$$

$$= 0.24192z^{4.3} + 0.195264z^{4.7} + 0.190253z^{4.8} + 0.02688z^{7.17} + 0.021696z^{7.83}$$

$$+ 0.021139z^8 + 0.06048z^{10.75} + 0.048816z^{11.75} + 0.047563z^{12}.$$

The probability that the system can produce the correct output (without respect to the task execution time) is:

$$R(\infty) = 0.24192 + 0.195264 + 0.190253 + 0.02688 + 0.021696$$

$$+ 0.021139 + 0.06048 + 0.048816 + 0.047563 = 0.854011$$

The probability that the system produces the correct output in time less than 10 seconds is:

$$R(10) = 0.24192 + 0.195264 + 0.190253 + 0.02688 + 0.021696 + 0.021139 = 0.697152.$$

The conditional expected system execution time is

$$W = (0.24192*4.3 + 0.195264*4.7 + 0.190253*4.8 + 0.02688*7.17 + 0.021696*7.83$$

$$+ 0.021139*8 + 0.06048*10.75 + 0.048816*11.75 + 0.047563*12) / 0.854011$$

$$= 6.086.$$

Example 2

Consider now a fault-tolerant software system with $n = 5$ running on hardware system consisting of 6 PUs. The availability and processing speed of each PU are presented in Table 3. The reliability and computational complexity of software versions are presented in Table 4.

Table 3 Parameters of PUs given in Example 2

No. of PU j	1	2	3	4	5	6
a_j	0.75	0.78	0.90	0.87	0.92	0.81
x_j	14.0	12.0	8.0	8.0	8.0	6.0

Table 4 Parameters of software versions for Example 2

Version i	1	2	3	4	5
r_i	0.7	0.6	0.8	0.6	0.9
c_i	46.0	57.0	70.0	72.0	83.0

The minimal possible execution time $\theta_{\min}$ and the indices $R(\infty)$ and W obtained for this system for different m are presented in Table 5. The corresponding functions $R(\theta^*)$ are presented in Fig. 2.

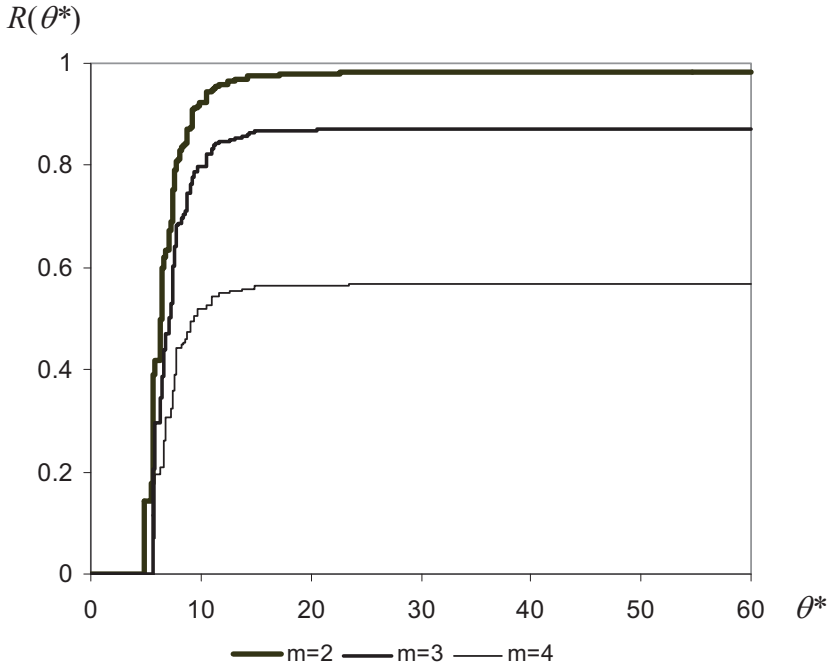

Fig. 2 System reliability functions $R(\theta^*)$ for different values of m

The maximum possible execution time is the same as $\theta_{max} = 54.67$ for all of the obtained solutions. It is equal to the time of execution of all of the versions on single slowest PU.

Table 5 System performance indices for different m

	θ_{min}	$R(\infty)$	W
$m = 2$	4.9	0.981	6.796
$m = 3$	5.59	0.871	7.270
$m = 4$	5.67	0.567	7.391

The system performance analysis allows one to estimate the influence of individual processing units on the overall system performance. For example, one can compare the system performance with and within a particular PU. Consider the software system with $m = 2$. The performance indices obtained for this system running on hardware system consisting of $N = 6$ PUs (all of the PUs are included into the system), $N = 5$ (the fastest PU is removed) and $N = 4$ PUs (two fastest PUs are removed) are presented in Table 6 and the corresponding functions $R(\theta^*)$ are presented in Fig. 3.

Table 6 System performance indices for different N

	θ_{min}	$R(\infty)$	W
$N = 6$	4.9	0.981429	6.796
$N = 5$	6.53	0.981397	8.785
$N = 4$	9.14	0.981246	11.860

Note that the difference in $R(\infty)$ in the obtained solutions is negligibly small. Indeed, this difference depends on the probability of simultaneous failure of all of the PUs, which is very small in the considered system. On the contrary, the system reliability $R(\theta^*)$ for any given allowed execution time $\theta^* < \theta_{max}$ is to much extent influenced by the structure of the hardware system (which can be seen in Fig. 3).

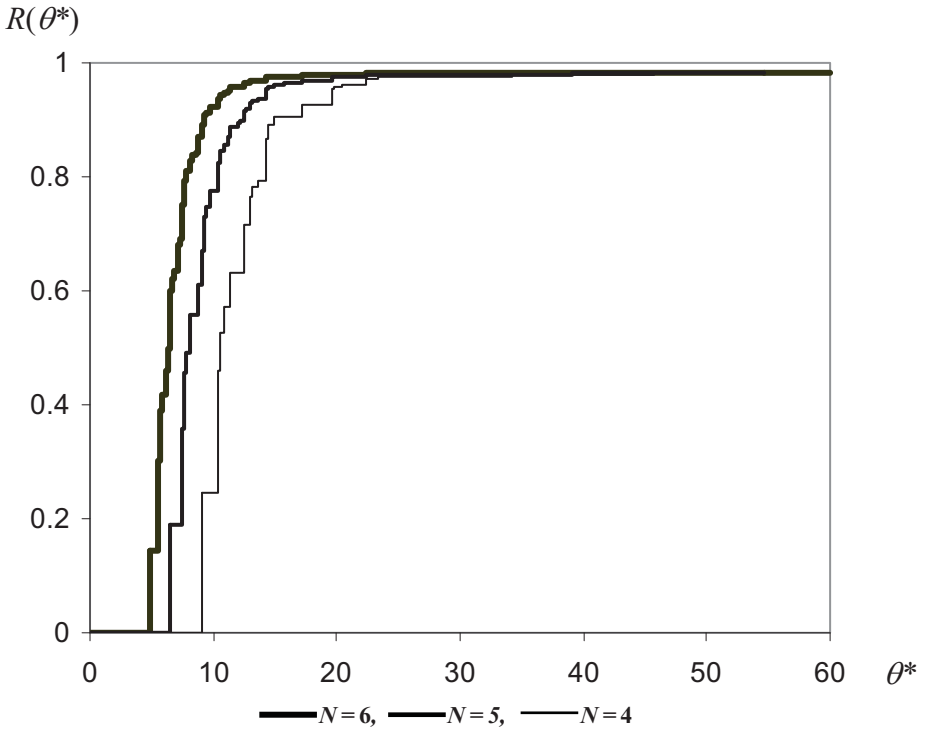

Fig. 3 System reliability functions $R(\theta^*)$ for different values of N

5. Conclusions

The suggested algorithm allows analyst to evaluate the expected performance (task execution time) and reliability (probability that the task is completed within a given time) of complex fault-tolerant hardware-software system consisting of nonidentical hardware components. It takes into account both reliability of software versions and availability of hardware units. The algorithm based on universal generating function technique calculates the indices in a negligible time, which provides possibility of fast comparison of different system structures, performing sensitivity analysis and optimization. Further research can be devoted to incorporation of common cause failures and imperfect work-sharing into the model, optimal scheduling of version execution in which limited hardware availability is taken into account and analysis of systems that consist of versions with random computational complexity.

Acknowledgement:

This research was carried out while the first author was visiting National University of Singapore supported by the research grant R-266-000-020-112 at National University of Singapore. The authors would like to thank two anonymous referees for several constructive comments on an earlier version of the paper.

References:

[1] Blazewicz J., Dell'Olmo P., Drozdowski M., Maczka P. Scheduling multiprocessor tasks on parallel processors with limited availability. *European Journal of Operational Research*, **149**(2), 2003, pp. 377-389

[2] Bertossi A., Fusiello A. Rate-monotonic scheduling for hard-real-time systems. *European Journal of Operational Research*, **96**(3), 1997, pp. 429-443.

[3] Drozdowski M. Scheduling multiprocessor tasks — An overview. *European Journal of Operational Research*, **94**(2) 1996, pp. 215-230.

[4] Levitin G., Optimal version sequencing in fault-tolerant programs, *Asia-Pacific Journal of Operational Research*, **22**(1), 2005, pp. 1-18.

[5] Czarnowski J ., Jedrzejowicz P ., Ratajczyk E., Scheduling fault-tolerant programs on multiple processors to maximize schedule reliability, in M.Felici, K.Kanoun, A.Pasquini (Eds.) *Computer Safety, Reliability and Security*, Lecture Notes in Computer Science 1698, 1999, p. 385-395.

[6] Czarnowski I., Jedrzejowicz P., Artifical Neural Network for Multiprocessor Tasks Scheduling. *Intelligent Information Systems*. Proceedings of the IIS'2000 Symposium*,* Bystra, Poland, 2000, p. 207-216.

[7] Jedrzejowicz P ., Wierzbowska I., Scheduling multiple variant programs under hard real-time constraints, *European Journal of Operational Research*, 127, 2000, pp. 458-465.

[8] Foster, I., Kesselman, C. and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations, *International Journal of High Performance Computing Applications,* 2001, 15, pp. 200-222.

[9] Nabrzyski, J., Schopf, J.M., Weglarz, J., *Grid Resource Management*, Kluwer Publishing, 2003.

[10] Wallner S., A Configurable System-on-Chip Architecture for Embedded and Real-Time Applications: Concepts, Design and Realization*, Elsevier Journal of Systems Architecture,* 51(6-7), pp. 350-367

[11] Teng X., Pham H., Software fault tolerance, in *Reliability Engineering Handbook,* H. Pham (Ed.), Springer, 2003, pp. 585-611.

[12] Bowles J.B., Dobbins J.G., Approximate reliability and availability models for high availability and fault-tolerant systems with repair. *Quality and Reliability Engineering International*, **20** (7), 2004, pp. 679-697.

[13] Scherrer C., Steininger A., Dealing with dormant faults in an embedded fault-tolerant computer system. *IEEE Transactions on Reliability*, **52** (4), 2003, pp. 512-522.

[14] Choi M., Park N., Lombardi F., Modeling and analysis of fault tolerant multistage interconnection networks, *IEEE Transactions on Instrumentation and Measurement,* **52** (5), 2003, 1509-1519.

[15] Fang L., Lu X.C., A cost effective fault-tolerant scheme for RAIDs. *Journal of Computer Science and Technology*, **18** (2), 2003, 230-234.

[16] Philippi S., Analysis of fault tolerance and reliability in distributed real-time system architectures, *Reliability Engineering and System Safety*, **82**(2), 2003, pp. 195-206.

[17] Littlewood B., Popov P., Strigini L., Assessing the reliability of diverse fault-tolerant software-based systems. *Safety Science*, **40** (9), 2002, pp. 781-796.

[18] Teng X.L., Pham H., A software-reliability growth model for N-version programming systems. *IEEE Transactions on Reliability*, **51** (3), 2002, pp. 311-321.

[19] Carrasco J.A., Computationally efficient and numerically stable reliability bounds for repairable fault-tolerant systems. *IEEE Transactions on Computers*, **51** (3), 2002, pp. 254-268.

[20] Berman O., Kumar U. Optimization models for recovery block schemes. *European Journal of Operational Research*, **115**(2), 1999, pp. 368-379.

[21] Tai A., Meyer J., Avizienis A., Performability enhancement of fault-tolerant software, *IEEE Transactions on Reliability,* **42** (2), 1993, pp. 227-237.

[22] Goseva-Popstojanova K., Grnarov A., Performability Modeling of N Version Programming Technique, *Proc. 6th IEEE International Symposium on Software Reliability Engineering* (*ISSRE'95*), Toulouse, France, Nov. 1995.

[23] Levitin G., Optimal structure of fault-tolerant software systems, *Reliability Engineering and System Safety*, **89**, 2005, pp. 286-295.

[24] Levitin G., Reliability and performance analysis for fault-tolerant programs consisting of versions with different characteristics, *Reliability Engineering & System Safety*, **86,** 2004, pp. 75-81.

[25] Chen L., Avizienis A., N-version programming: a fault tolerance approach to the reliable software, *Proc. 8th Int. Sym. Foult-Tolerant Computing*, Toulouse, France; 1978, pp. 3-9.

[26] Randell B., System structure for software fault tolerance, *IEEE Transactions on Software Engineering,* **1**(2), 1975, pp. 220-232.

[27] Meyer J. On evaluating the performability of degradable computing systems, *IEEE Transactions on Computers,* **29**, 1980, pp. 720-731.

[28] Grassi V., Donatiello L., Iazeolla G., Performability evaluation of multicomponent fault-tolerant systems, *IEEE Transactions on Reliability,* **37** (2), 1988, pp. 216-222.

[29] Ushakov I., Optimal standby problems and a universal generating function, *Soviet Journal of Computer Systems Science*, **25**, 1987, pp. 79-82.

[30] Levitin G., Lisnianski A., Beh-Haim H., Elmakis D., Redundancy Optimization for Series-parallel Multi-state Systems, *IEEE Transactions on Reliability*, **47**, 1998, pp. 165-172.

[31] Levitin G. *Universal generating function in reliability analysis and optimization,* Springer-Verlag, London, 2005.

[32] Levitin G., Xie M., Performance distribution of fault-tolerant programs in the presence of positive failure correlation. Submitted to *IIE Transactions*.