Информатика. Экономика. Управление//
Informatics. Economics. Management

2023; 2(3)
http://oajiem.com/

УДК: 004.942
DOI: https://doi.org/10.47813/2782-5280-2023-2-3-0201-0209

EDN: **FVKPOM**

Improving decision-making algorithms on the correctness of the states of multiversions of fault-tolerant software systems

D. I. Kovalev[1,2,3], P. K. Zaitsev[4]

[1]Krasnoyarsk Science and Technology City Hall, Krasnoyarsk, Russia
[2]Krasnoyarsk State Agrarian University, Krasnoyarsk, Russia
[3]National Research University "Tashkent Institute of Irrigation and Agricultural Mechanization Engineers", Tashkent, Uzbekistan
[4]Siberian Federal University, Krasnoyarsk, Russia

Abstract. The task of improving the algorithms for making decisions about the correctness of the states of multiversions is relevant today for the development of fault-tolerant software systems. The article considers a class of decision-making algorithms that are implemented in the execution environment of multi-version program modules. This execution environment includes a decision block, which, based on the principle of voting, allows you to get the correct decision at the output, filtering out the erroneous results of the execution of software versions. Advanced algorithms allocate the output results of multiversion triggering into classes or subsets, which are then analyzed for correctness. This approach is characterized by the fact that it allows you to increase the stability of the runtime environment to interversion errors. This, in turn, helps to improve the fault tolerance of software systems used in reliability-critical applications.

Keywords: software, algorithm, reliability-critical application, fault-tolerant system, decision-making.

For citation: Kovalev, D. I., & Zaitsev, P. K. (2023). Improving decision-making algorithms on the correctness of the states of multiversions of fault-tolerant software systems. Информатика. Экономика. Управление - Informatics. Economics. Management, 2(3), 0201–0209. https://doi.org/10.47813/2782-5280-2023-2-3-0201-0209

INTRODUCTION

The software and hardware environments for the execution of multiversions that are part of fault-tolerant software systems, as a rule, include a decision block. This block functions on the basis of multiversion voting algorithms. Voting algorithms are designed to separate the outputs of multiversions. The outputs of multiversions are correctly assigned to two classes.

The first class includes correct outputs, the second class includes incorrect outputs [1-4]. Thus, an important task is to separate or classify the outputs of multiversions, which is the subject of quite a lot of work in the field of multiversion programming for fault-tolerant software systems [5-7]. Traditionally, well-known voting algorithms are used, which include: voting by absolute majority, voting by consensus and voting by fuzzy consensus [6].

The improvement of these algorithms is possible by introducing weight coefficients for multiversions, which makes it possible to treat the outputs of multiversions with a higher coefficient in real time, taking into account changes in these weight coefficients. This allows us to switch to the use of weighted consensus voting and fuzzy weighted consensus voting algorithms. In this case, after comparing the outputs of multiversions, it becomes possible to group them according to the similarity of outputs into several classes or subsets of multiversions. However, there is the following limitation of this approach. It lies in the fact that in the case when the output of the multiversion is characterized not by an integer, but has a fractional value, it is difficult to establish the identity of the outputs of the multiversions. It is proposed to introduce the concept of equality relation. Based on this ratio, we can conclude that two numbers are equal if they differ from each other by an amount that is less than some tolerance.

Note that the equality relation does not satisfy the transitivity condition. That is, from the fact that $|a - b| < \varepsilon$ and $|b - c| < \varepsilon$, it does not follow that $|a - c| < \varepsilon$, where a, b and c are some numbers. However, there is always the danger of misclassification, which can potentially lead to incorrect voting results of multiversions and incorrect decision-making about the correctness of their output. This can significantly affect the operation of the software package and the failure of the control system for objects that are critical in terms of reliability, which is noted by many authors, for example, in [8-11].

We also note that a number of authors are supporters of methods that make decisions by voting without classifying the outputs of multiversions. These are, for example, median voting and weighted median voting [12]. The essence of these algorithms is that the average value of all results is chosen and this average value is recognized as the correct result of multiversion voting.

ABSOLUTE MAJORITY VOTING ALGORITHM

The Absolute Majority Voting Algorithm (AMVA) is the simplest decision-making algorithm in multiversion systems. When voting by an absolute majority, in order to make a

decision, it is necessary that at least $m = \left\lceil \frac{N+1}{2} \right\rceil$ versions have identical results (where $\lceil \ \rceil$ is the rounding operator to the nearest higher integer). For example, for a multi-version system with 7 independently developed and functionally equivalent versions, the threshold m would be 4.

It is believed that the outputs of the majority (i.e., the class with the number of elements equal to or greater than m) of the multiversions are correct, and the rest are erroneous. If there are fewer equal outputs than m, then this situation is treated as uncertainty and it is impossible to make a decision.

The disadvantage is that in practice there are not rare cases when the size of any of the classes does not exceed m. In addition, it is possible that most multiversions return an erroneous result, and it will be treated as correct. This, in turn, can also lead to system failure.

The advantage of this algorithm is its simplicity and minimal memory and resource requirements. With a low degree of interversion errors, this algorithm is quite effective. However, in practice it is rarely possible to achieve such a situation. An interversion error is understood as a situation when several multiversions on the same set of input data return erroneous outputs (not necessarily equal to each other).

To implement this method, a Boolean matching matrix *R* is used. The elements of this matrix are defined as follows:

$$r_{ij} = \begin{cases} 1 & |x_i - x_j| < \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

In other words, if the output of version i "agrees" with the output of version j and additionally includes some area ε, then $r_{ij} = 1$, otherwise $r_{ij} = 0$.

To make a decision using the absolute majority voting algorithm, it is necessary to find a row i in the matrix *R*, the sum of whose elements will be greater than or equal to the threshold value m. Then all multiversions j, which correspond to r_{ij} equal to one, are correct, and the rest are erroneous. If no such string exists, an error occurs. In this case, no decision can be made.

AGREED MAJORITY VOTING ALGORITHM

To make a decision using the agreed majority voting algorithm (AGMVA), it is necessary to choose such a class of multiversion outputs, the number of elements in which is greater than in all the others (i.e., the class with the maximum number of elements). If there are several such classes, a random choice of any of them is proposed.

For implementation, as in the AMVA, a matching matrix *R* is compiled. The choice of the required set of correct outputs is carried out in two stages:

- all rows are selected and stored, the sum of elements of which is maximum;
- any of these rows is randomly selected.

If the found row has number *i*, then all multiversions *j*, which correspond to r_{ij} equal to one, are correct, and the rest are erroneous.

The main advantage of this algorithm, compared to absolute majority voting, is a higher resistance to inter-version errors. The disadvantages are the presence of an element of randomness in the decision-making process and a higher memory requirement. For example, if there are two equal classes, the probability of choosing the correct output is 0.5.

The main disadvantage of the classical AGMVA-algorithm is that the decision is made without taking into account the accumulated comparison statistics. If the reliability of the program modules themselves is not high enough, the probability that a class with a relatively large number of elements will be correct is reduced. This is due, on the one hand, to the fact that correct states can also be in a group with a slightly smaller number of elements than the largest one. And on the other hand, this is an element of randomness when choosing among classes with the same number of states.

As an increase in resistance to interversion errors, and an increase in the efficiency of the classical technique as a whole, it can be proposed to make a decision not based on the very fact of the majority of equal states, but based on the weight of the class. By the weight of a set of outputs, we mean the sum of the probabilities of a correct output of the multiversions included in it and the sum of the probabilities of an erroneous output for those not included:

$$w_i = \sum_{j=1}^{n} \begin{bmatrix} p_j & r_{ij} = 1 \\ 1 - p_j & r_{ij} \neq 1 \end{bmatrix},$$

where:

- *i* is the row number in the matching matrix;
- p_j is the probability of correct output of multiversion *j*;
- *n* is the number of multiversions.

As p_j, you can use the accumulated statistical information about the progress of the multiversions.

Then the scheme for choosing the desired class of outputs takes the form:

- all rows with the maximum weight are selected and stored;
- any of these rows is randomly selected.

This scheme is called Weighted Consensus Majority Voting (WCMV). Due to the use of probabilities, this algorithm imposes a lower bound on the average reliability of software modules. The advantage of weighted voting is a very high resistance to inter-version errors and a decrease in the influence of the element of chance, compared with the basic version of the methodology.

FUZZY CONSENSUS VOTING ALGORITHM

The main problem of applying the equality relations is the strictly specified deviation of the output values, which does not allow comparisons with an accuracy exceeding ε. The use of fuzzy sets helps to solve this problem.

In crisp sets, an element may or may not be an element of the set. Let the degree of membership of an element in the set be given by some function, which we will call the *characteristic function*. Fuzzy sets contain elements that have different *degrees of membership*. The degree of belonging is given by a number on the interval from 0 to 1, the higher the degree, the closer the number is to 1.

In the classical theory of voting algorithms, equality relations based on crisp sets are used. It is considered that *x* is equal to some number *a* if |x - a| < ε.

Then the *characteristic function* of this equality can be represented as follows:

$$\lambda_X(x) = \begin{cases} 1 & x \in (a - \varepsilon/2\,; a + \varepsilon/2) \\ 0 & x \notin (a - \varepsilon/2\,; a + \varepsilon/2) \end{cases}.$$

This function has a rectangular shape (Figure 1a) with a center at point a and a base width equal to ε.

The *degree of membership* inside this rectangle is 1, outside it is 0. However, it is more convenient to set the degree of membership so that it decreases as you move away from *a*. For example, set it in the form of a triangle (Figure 1b):

$$\mu_{\tilde{X}}(x) = \begin{cases} 1 - \frac{|a-x|}{\varepsilon/2} & x \in (a - \varepsilon/2\,; a + \varepsilon/2) \\ 0 & x \notin (a - \varepsilon/2\,; a + \varepsilon/2) \end{cases}.$$

The matching matrix in this case will be defined as $R = \{r_{ij}\}$, where

$$r_{ij} = \mu_{\tilde{R}}(x_i, x_j) = \begin{cases} 1 - \frac{|x_i - x_j|}{\varepsilon} & |x_i - x_j| < \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

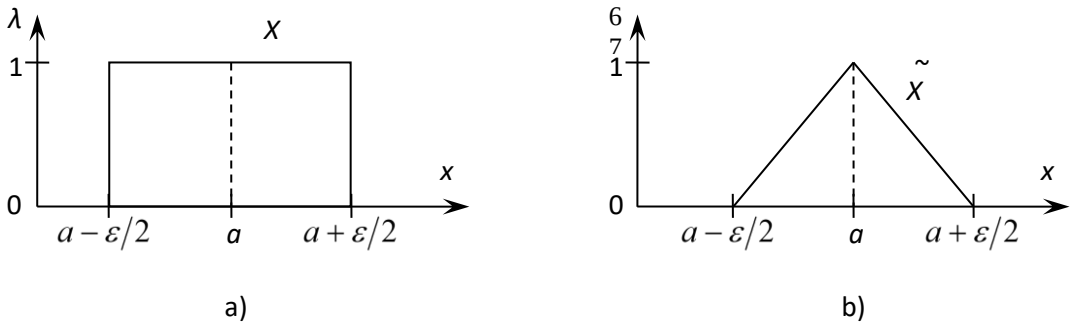

Figure 1. The characteristic function of the ratio of equality *x* and *a*:
a) a clear set; b) a fuzzy set in the form of a triangle.

Next, the values of the elements of the resulting matrix *R* are truncated to the Boolean form $\bar{R}$ according to some given threshold value *λ*. Moreover, the value of *λ* is often indicated in the name of the technique used. For example, for λ=0.5, the name of algorithm would be FCVA-0.5.

Truncation of values occurs according to the following scheme:

$$\bar{r_{ij}} = \begin{cases} 1 & r_{ij} \geq \lambda \\ 0 & r_{ij} < \lambda \end{cases}$$

After that, the resulting matrix $\bar{R} = \{ \bar{r_{ij}} \}$ is used as the consensus matrix in the clear consensus voting method (both basic and weighted) discussed above.

MEDIAN VOTE

In this section, for comparison with the improved multiversion voting algorithms that were presented in the article, we consider an approach that is based on the median voting algorithm. The starting position is that all outputs of multiversions are recognized as incorrect, that is, they are considered erroneous. As a correct result, the decision block chooses the average between them. The coefficient for median voting is usually determined as follows:

$r = \frac{1}{n} \sum_{i=1}^{n} \alpha_i \cdot x_i$, where:

- *r* is the correct result;
- α_i is the weighting factor;
- x_i is the output of multiversion *i*;
- *n* is the number of multiversions.

If the weighting factor $\alpha_i \neq 1$ then this algorithm is called *weighted* median voting.

These algorithms are used in cases where it is difficult to directly compare the results of multiversions. For example, when a multiversion system is used to find optimal directions. In such a situation, it makes no sense to directly compare the vectors, since they are at least of different lengths.

CONCLUSION

The improved voting algorithms considered in the article make it possible to unify the operation of the decision block when implementing the multiversion execution environment of various software systems for fault-tolerant applications. The use of improved algorithms expands the possibilities of using multiversion programming implemented on the basis of other well-known models [12-15]. In these works, in particular, such models as the recovery block model, the sequentially recovering blocks model, N-version programming, t/(n-1)-version programming, and N-version programming with self-checking are implemented.

For the universal multiversion execution environment, which includes a set of voting algorithms considered in this article, a comparative voting technique has been developed. This technique allows us to evaluate the effectiveness of the models listed above. For each model, when synthesizing the decision block, a reasonable choice of the most appropriate voting algorithm is carried out. Note that improved algorithms for multiversion voting make it possible to increase the stability of the decision block against interversion errors. It is these errors that significantly affect the fault tolerance of the software package used to control control systems that are critical in terms of badness and operate in real time.

REFERENCES

1. Kovalev I., Losev V., Saramud M., Petrosyan M. Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects. MATEC Web of Conferences. 2017; 132: 04011. https://doi.org/10.1051/matecconf/201713204011

2. Kovalev I.V., Zavyalova O.I., Laikov A.N. Formation of redundant software for fault-tolerant control systems. News of higher educational institutions. Instrumentation. 2008; 51(10): 30-34.

3. Kovalev I.V., Slobodin M.Ju., Tsarev R.Ju. Multi-version design of fault-tolerant software in control systems. Engineering & Automation Problems. 2006. 1: 61-69.

4. Kulyagin A. V., et al. N-version design of fault-tolerant control software for communications satellite system. 2015 International Siberian Conference on Control and Communications. SIBCON 2015 Proceedings, Omsk, 21-23 May 2015. Omsk; 2015: 7147116. https://doi.org/10.1109/SIBCON.2015.7147116

5. Kotenok A.V. Implementation of multiversion voting algorithms. Bulletin of the University Complex. 2004; 3(17): 86 93.

6. Kotenok A.V. Environment for multiversion execution of program modules. Bulletin of the University Complex. 2006; 6(20): 219 238.

7. Kovalev I.V., Kotenok A.V. On the problem of choosing a decision-making algorithm in multiversion systems. Information Technology. 2006; 9:39-44.

8. Kovalev I.V., Slobodin M.Yu., Tsarev R.Yu. Taking into account the subjective preferences of the decision maker in the multiversion design of automated control systems. Control systems and information technologies. 2005; 1(18): 44-49.

9. Saramud M.V., Kovalev I.V., Losev V.V., Petrosyan M.O. Software interfaces and decision block for the execution environment of multi-version software in real-time operating systems. International Journal on Information Technologies and Security. 2018; 10(1): 25-34.

10. Engel E. A., Kovalev I. V. Information processing using intelligent algorithms by solving WCCI 2010 tasks. Siberian Aerospace Journal. 2011; 3(36): 4-8.

11. Kovalev I.V., Zelenkov P.V., Karaseva M.V., Tsarev M.Y., Tsarev R.Y. Model of the reliability analysis of the distributed computer systems with architecture "Client-Server". IOP Conf. Ser.: Mater. Sci. Eng. 2015; 70: 012009. https://doi.org/10.1088/1757-899X/70/1/012009

12. Kovalev I.V. Analysis of problems in the field of software reliability research: multi-stage and architectural aspect. Siberian Aerospace Journal. 2014; 3(55): 78-92.

13. Kovalev I.V., Kovalev D.I., Ambrosenko N.D., Borovinsky D.V. Analysis of test problems of multiversion formation of fault-tolerant software systems. Modeling, optimization and information technology. 2022; 10(2). https://doi.org/10.26102/2310-6018/2022.37.2.003

14. Kovalev I.V., Nova A.V., Shtentsel A.V. Estimation of the reliability of the multiversion software architecture of control and information processing systems. Siberian Aerospace Journal. 2008; 3(20): 50-52.

15. Gruzenkin, D., Shavarin, D. Recovery blocks method to improve software reliability: comparison with N-version programming. Modern Innovations, Systems and Technologies. 2022; 2(3): 0127–0138. https://doi.org/10.47813/2782-2818-2022-2-3-0127-0138

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Ковалев Дмитрий Игоревич, аспирант, кафедра Информационных технологий и математического обеспечения информационных систем, Красноярский государственный аграрный университет, Красноярск, Россия
e-mail: grimm7jow@gmail.com
ORCID: https://orcid.org/0000-0003-2128-6661

Dmitry Kovalev, PhD student, the Department of Information Technologies and Software for Information Systems, Krasnoyarsk State Agrarian University, Krasnoyarsk, Russia

Зайцев Павел Константинович, аспирант, кафедра Информатики, Сибирский федеральный университет, Красноярск, Россия
e-mail: pkz97@mail.ru
ORCID: https://orcid.org/0000-0002-6564-2325

Pavel Zaitsev, PhD student, the Department of Informatics, Siberian Federal University, Krasnoyarsk, Russia

Статья поступила в редакцию 05.06.2023; одобрена после рецензирования 10.07.2023; принята к публикации 12.07.2023.

The article was submitted 05.06.2023; approved after reviewing 10.07.2023; accepted for publication 12.07.2023.