Classification of voting algorithms for N-version software

IOP Conf. Series: Journal of Physics: Conf. Series **1015** (2018) 042060 doi:10.1088/1742-6596/1015/4/042060

Classification of voting algorithms for N-version software

R Yu Tsarev[1], **M S Durmuş**[2], **I Üstoglu**[3], **V A Morozov**[1]

[1] Siberian Federal University, 79, Svobodny pr., Krasnoyarsk, 660041, Russia
[2] Pamukkale University, Kinikli Campus, Denizli, 20070, Turkey
[3] Yıldız Technical University, Davutpaşa Mah., Esenler, Istanbul, 34220, Turkey

E-mail: tsarev.sfu@mail.ru

Abstract. A voting algorithm in N-version software is a crucial component that evaluates the execution of each of the N versions and determines the correct result. Obviously, the result of the voting algorithm determines the outcome of the N-version software in general. Thus, the choice of the voting algorithm is a vital issue. A lot of voting algorithms were already developed and they may be selected for implementation based on the specifics of the analysis of input data. However, the voting algorithms applied in N-version software are not classified. This article presents an overview of classic and recent voting algorithms used in N-version software and the authors' classification of the voting algorithms. Moreover, the steps of the voting algorithms are presented and the distinctive features of the voting algorithms in N-version software are defined.

1. Introduction

One of the main approaches for the development of highly reliable and fault-tolerant software is the N-version programming (NVP) method. This method was proposed in 1977 by Avizienis and Chen [1]. According to the idea of the N-version programming, *N* different software versions should be developed. These versions have the same input-output specifications and they are functionally equivalent, i.e. they solve the same task. However, the methods for task solving are different.

The main advantage of this method is to provide software fault tolerance [2]. Even in the case of failure of any individual version, the rest of the running versions will produce their results, and the system will continue to function. In this case, the usual functioning of the system is also insured against residual errors that were not identified in the testing phase [1].

The N-version programming has shown its efficientcy in a variety of control and information processing systems in such safety-critical areas as space systems, message passing systems, railway interlocking systems, plagiarism detection, web services, see e.g., [2-7].

As mentioned before, the N-version programming method involves the generation of independent and parallel execution of several versions of the same software module. All versions of the module obtain the same input data. Since there is a possibility that the versions will produce different results of calculations due to design diversity, an important problem which is the determination of the valid and the wrong results arises. The outputs of *N* different software versions are sent to another unit which is known as the voter. The voter collect the outputs of *N* different software versions and produces the final decision depending on a voting algorithm. The voting algorithm determines which result is true, and transmit its decision to the input of the next module or return its decision to the user. Thus, voting algorithm is the most important component of N-version software.

There are many algorithms for voting, which differ in schemes of work and the requirements of the original data, see e.g., [8-13]. Each algorithm has its own advantages and disadvantages, and there are certain conditions for its successful application. Furthermore, depending on the provided set of data by the versions, some algorithms may be ineffective or may not provide an opportunity to make a decision on the correctness of outputs. Among the entire set of voting algorithms there is a series of algorithms that have common features. The aim of this study is to provide a general classification of voting algorithms that can be applied in the N-version software. An overview and classification of the voting algorithms used in N-version software is given below.

2. Voting algorithms which are based on the comparison of the output data

Before proceeding to the consideration of this class of algorithms, the main features of their application are noted as follows:

1) The heart of the application of the algorithms relies on the assumption that the probability of identical-and-wrong answers is negligible [9]. Identical-and-wrong answers are those which arise when multiple versions produce erroneous results.

2) The whole set of the results, returned by the versions, is divided into a subset of “correct” and a subset of “incorrect” outputs. Selection of a set of “correct” answers is based on a comparison of the entire set of results.

3) When comparing the results of the versions, the concept of the equivalence of the outputs is used. Thus, the output of one version is equivalent to the output of another version, if the output values differ by no more than some fixed number which is called as the tolerance value.

4) As a rule, a set of equivalent outputs is accepted as a correct result. This set is selected from the entire set of the results of the versions.

5) Choosing the correct set of the results of the versions is done either by using agreement matrix or by using subsets of agreeing versions.

Voting algorithms, based on the comparison of the output data, can be divided into two subclasses: formalized and non-formalized voting algorithms.

2.1. Non-Formalized voting algorithms

Non-formalized voting algorithms differ from formalized in the following feature: during the analysis, the whole set of the output data (results) of the versions is divided into disjoint subsets.

It is convenient to consider the algorithms of this subclass by dividing them into several groups:

1. Classic algorithms.
2. Fuzzy algorithms.
3. Minimization algorithms.

Classic voting algorithms include the N-version programming with the majority voting algorithm (NPV-MV) and N-version programming with the consensus voting algorithm (NVP-CV). These algorithms are discussed in a veriety of papers such as [7], [8], [12-18].

Fuzzy voting algorithms are based on the application of the theory of fuzzy logic. The fuzzy voting algorithms in N-version software are studied in papers [16], [19-22]. The fuzzy consensus voting algorithm (Fuzzy CV) and the fuzzy majority voting algorithm (Fuzzy MV) belong to this group.

Since the first two groups of the voting algorithms are well-known, there is no need to discuss them in detailed in this paper. The peculiarity of the 3rd group – the voting algorithms with minimization – is that the multitude of the outputs is divided into disjoint subsets of equivalent elements without using the compositions and the achieved subsets cannot intersect. As long as an output is inside of any subsets, it is no longer counted and only the remaining multitude of outputs is considered. Voting algorithms with minimization are described in papers [8], [13], [16].

2.1.1. The majority voting algorithm with minimization (minMV).

Let us assume that the number of version of a module of N-version software is equal to N and the multitude of the outputs of the versions are denoted by the set $X = (x_1, x_2, \ldots, x_N)$. After setting tolerance value ε, the majority voting algorithm with minimization consists of the following steps:

Step 1. Partition of multitude set X into subsets. This step implies the following operations:

Step 1a. Select random x_i value from set X and put this value into subset C_i. Then, put all the other values from multitude set X into subset C_i which are consistent with the previously chosen x_i value, or in other words, the other multitude values with a difference from multitude value x_i as ε.

Step 1b. Remove all elements of subset C_i from multitude set X.

Step 1c. If multitude set X contains at least one element, then go to Step 1a. Otherwise, proceed to Step 2.

Step 2. Determine the correct set of outputs.

At this step, the obtained subsets are considered where the number of elements is counted for each subset. Let us denote the number of elements in i-subsets as Yc_i. If there is subset C_i for which

$$Yc_i \geq \left\lceil \frac{N+1}{2} \right\rceil \quad (1)$$

is satisfied, then the number of correct outputs is defined as subset C_i. If a subset which satisfies the inequality (1) does not exist, then making a decision by using the minMV algorithm is impossible.

2.1.2. The consensus voting algorithm with minimization (minCV).

Let us consider the same initial problem statement, where N versions' results represented by the set of outputs $X = (x_1, x_2, \ldots, x_N)$. The consensus voting algorithm with minimization consists of the following steps.

Step 1 is equvalent to step 1 of the majority voting algorithm with minimization.

Step 2. Determine the correct set of outputs.

At this step, the obtained subsets are considered where the number of elements is counted for each subset. Let us denote the number of elements in i-subsets as Yc_i. Then, the subdivision with maximum Yc_i value is selected and the set of outputs is defined as subdivision C_i. If there are several subdivisions with the maximal number of elements, a random multitude (one of those with the maximal number of items) is selected as the set of correct outputs.

Nevertheless, the minCV algorithm produces a result even if there are no consistent versions and the minCV algorithm returns the randomly selected output.

2.2. Formalized voting algorithms

Formalized voting algorithms have particular importance since they allow splitting the set of the multitude of the outputs into subdivisions during the voting process. Those subdivisions may intersect, which indicate that in the classification, some outputs can be included into more than one subdivision [16], [19]. The FMV and the FCV algorithms belong to this class of algorithms.

2.2.1. The formalized majority voting algorithm (FMV).

Let us assume that the number of version of any module of N-version software is equal to N where $x_1, x_2, \ldots, x_N$ are the values of outputs. After setting tolerance value ε, the FMV algorithm consists of the following steps:

Step 1. Partition of the multitude of outputs into subdivisions.

The multitude of versions is split into N subdivisions. Each output is characterized with its subdivision. Subdivision C_i is developed for output i ($i = 1, \ldots, N$) as follows:

$$C_i = \{x_j \mid \forall j, \mid x_i - x_j \mid \leq \varepsilon\},\ j = 1,\ldots, N.$$

In this case each output belongs to at least one subdivision.

Step 2. Determine the correct set of outputs.

At this step, the obtained subsets are considered where the number of elements is counted for each

subset. Let us denote the number of elements in *i*-subsets as Yc_i. If there is subdivision C_i for which (1) is satisfied, the set of correct outputs is defined as subdivision C_i. If there is no subdivision which satisfies requirements (1), making a decision with the FMV algorithm is impossible.

2.2.2. The formalized consensus voting algorithm (FCV).

Let us consider the same initial problem statement, where *N* versions' results are represented by the set of outputs $X = (x_1, x_2, \ldots, x_N)$. The FCV algorithm consists of the following steps:

Step 1 is equvalent to step 1 of the FMV algorithm.

Step 2. Determine the correct set of outputs.

At this step the obtained subsets are considered where the number of elements is counted for each subset. Let us denote the number of elements in *i*-subsets as Yc_i. Then, the subdivision with maximum Yc_i value is selected and the set of outputs is defined as subdivision C_i. If there are several subdivisions with the maximal number of elements, a random multitude is selected as the set of correct outputs.

Nevertheless, the FCV algorithm produces a result even if there are no consistent versions and the FCV algorithm returns the randomly selected output.

3. Voting algorithms in which the decision making does not depend on the similarity of the output data

The main characteristic feature of these algorithms is that the result of their evaluation does not depend on how the outputs of the versions are compromised. The maximum likelihood voting algorithms (MLV) and the average voting algorithms are included in this class of algorithms.

3.1. The maximum likelihood voting algorithm (MLV)

The MVL algorithm is proved to be the one of the most reliable ways of decision making in N-version software, see e.g., [9], [23], [24].

In consideration of the MLV algorithm, the following mathematical notations will be used:

- N – the number of functionally equivalent versions;
- m – the potency of outputs set;
- p_i – probability that version i return a correct result;
- V_i – variable that represents output of *ith* version;
- v_i – value of V_i for each output (one of m possible);
- V_{cor} – correct output; $\Pr\{x\}$ – probability of event x.

The following assumptions lie in the basis of the MLV algorithm [9]:

- Failures of functionally equivalent version of N-version software are not statistically dependent.
- Potency of the multitude of possible answers is defined prior to the voting and is equal to m.
- There is no multitude of correct answers. Only one output from the out of the entire multitude set is correct.
- In case of version failure, the version produces one of $m - 1$ erroneous results.

Let version i has single correct state (produces correct output) and $m - 1$ erroneous states. Let us assume that the probability of the erroneous state of the version is equal to $(1 - p_i) / (m - 1)$. With ζ_j, we denote the event in which output j is correct. Then there are m mutually exclusive events ($j = 1, \ldots, m$). Probability of realization of a separate multitude from N outputs may be represented as follows:

$$\Pr\{V_1 = v_1, V_2 = v_2, \ldots, V_N = v_N\} = \sum_{j=1}^{m} \left[\Pr_j\{V_1 = v_1\} \times \Pr_j\{V_2 = v_2\} \times \ldots \times \Pr_j\{V_N = v_N\}\right],$$

where $\Pr_j\{V_i = v_i\}$ is defined as follows:

$$\Pr_j\{V_i = v_i\} = \begin{cases} p_i, & \text{when } v_i = j, \\ \dfrac{1 - p_i}{m - 1}, & \text{when } v_i \neq j. \end{cases} \quad (2)$$

Equation (2) defines the probabilities of two possible events: a) probability that j (one of the

possible outcomes) is a correct answer and the output of version *i* is equal to *j*; b) probability that *j* is a correct answer and the output of version *i* is equal to one of *m* – 1 possible outcomes. Now it is possible to figure out the conditional probability that output *j* is a correct answer for this multitude of outcomes:

$$\Pr\{V_{cor} = j \mid V_1 = v_1, V_2 = v_2, ..., V_N = v_N\} = \frac{\Pr_j\{V_1 = v_1\} \times \Pr_j\{V_2 = v_2\} \times ... \times \Pr_j\{V_N = v_N\}}{\sum_{j=1}^{m} \left[\Pr_j\{V_1 = v_1\} \times \Pr_j\{V_2 = v_2\} \times ... \times \Pr_j\{V_N = v_N\}\right]}. \quad (3)$$

The MLV algorithm implies the selection of the output which has the greatest conditional probability according to (3). If maximal probability corresponds to more than one output, the selection is done randomly.

3.2. The averaged voting algorithm

With the use of the averaged voting, the average value from the whole multitude set of the outputs is taken as a result [16]. This algorithm has one fundamental difference from all the others where "correctness" and "failure" notions are not considered and the output data are simply averaged. The algorithm may be used only in exceptional cases when the comparison of the outputs of the versions is impossible.

4. Classification of the voting algorithms

Having been analysed, the voting algorithms described above were divided into several classes. Figure 1 demonstrates the classification of the voting algorithms used in the N-version software.

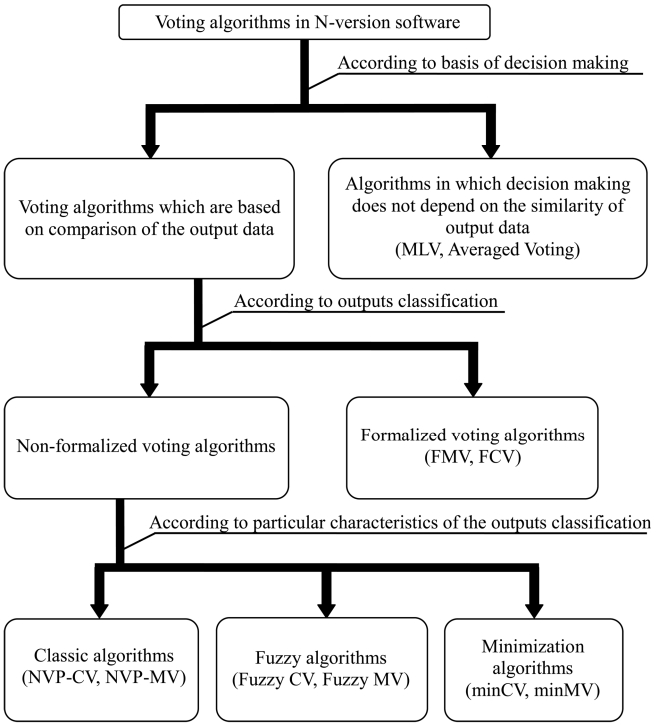

Figure 1. Classification of voting algorithms in N-version software.

Figure 1 shows that the classification is based on the decision making principle of the voting algorithms, on the classification of the output data and the particular characteristics of the output data classification.

5. Conclusion

N-version programming is an efficient and promising approach to increase the reliability and availability of the software. It strongly demands precise determination which versions returned correct results and which ones are failed. For this purpose, while the voting algorithms are being used, selecting the correct answer from the whole multitude set of the computation results is required.

Comparative analysis of the currently used algorithms showed that the most spread and versatile algorithms possess a number of common properties, which allowed putting them into a classification according to their characteristics. In the classification, proposed algorithms are divided according to their characteristics such as similarity tracking of the outputs, classification principle of the outputs and particular characteristics of the outputs.

Some algorithms produce a result anyway (MLV and NVP-CV), but some are used only in particular conditions (minMV and NVP-NV). Algorithms based on the comparison of versions output data are efficient and intuitive.

Analysis of the algorithms, conditions of their usage and the proposed classification in this paper tend to help practitioners, design engineers and software developers for N-version programming to choose and implement the algorithm that best fits the tasks they have during software development.

References

[1] Avizienis A and Chen L 1977 *Proc. Int. Conf. COMPSAC'77* (Chicago, IL) 149–155
[2] Avizienis A 2004 *Proc. 10th IEEE Pacific Rim Int. Symp. Depend. Comput.* (Papeete Tahiti, French Polynesia) 336
[3] Chernigovskiy A S, Tsarev R Y and Knyazkov A N 2015 *Proc. Int. Siberian Conf. on Control and Communications, SIBCON 2015* (Omsk, Russian Federation)
[4] Chitsaz B and Razzazi M 2012 *Proc. MIPRO 2012* (Opatija, Croatia) 345–348
[5] Eriş O, Yildirim U, Durmuş M S, Söylemez M T and Kurtulan S 2012 *Proc. CTS 2012* (Sofia, Bulgaria) 177–180
[6] Zhang F, Jhi Y-C, Wu D, Liu P and Zhu S 2012 *Proc. ISSTA 2012* (Minneapolis, MN) 111–121
[7] Looker N, Munro M and Xu J 2005 *Proc. COMPSAC 2005* (Edinburgh, Scotland, UK) 66–69
[8] Akhil K and Kavindra M 1991 *IEEE Trans. Reliab.* **40(5)** 593–600
[9] Kim K, Vouk M A and McAllister D F 1996 *Proc. ISSRE'96* (White Plains, NY) 330–339
[10] Latif-Shabgahi G, Bennett S and Bass J M 2003 *Microprocess. Microsyst.* **27(7)** 303–313
[11] Lin X, Yacoub S M, Burns J and Simske S J 2003 *Pattern Recogn. Lett.* **24(12)** 1959–1969
[12] Mohamed A and Zulkernine M 2007 *Proc. IEEE Int. Symp. High Assurance Systems Engineering* (Dallas, TX) 267–274
[13] Yacoub S 2003 *Reliab. Eng. Syst. Safe.* **81(2)** 133-145
[14] Ahamad M and Ammar M H 1989 *IEEE Trans. Softw. Eng.* **15(4)** 492–496
[15] Levitin G and Lisnianski A 2001 *Reliab. Eng. Syst. Safe.* **71(2)** 131–138
[16] Parhami B 1994 *IEEE Trans. Reliab.* **43(4)** 617–629
[17] Xie M and Pham H 2005 *Reliab. Eng. Syst. Safe.* **87(1)** 53–63
[18] Aghajan Z and Azgomi M A 2009 *Proc. IIT '09* (Al-Ain, United Arab Emirates) 304–308
[19] Kim K, Vouk M A and McAllister D F 1998 *Proc. IEEE Aerospace Conf.* (Aspen, CO) 5–19
[20] Hsu H-M and Chen C-T 1996 *Fuzzy Sets Syst.* **79(3)** 279–285
[21] Manic M and Frincke D 2001 *Proc. IECON'2001* (Denver, CO) 84–89
[22] Wang P 2007 *Proc. ICEMI'07* (Xi'an, China) 2666–2669
[23] Tamura S, Higuchi S andTanaka K 1971 *IEEE Trans. Syst. Man Cybern.* **1(1)** 61–66
[24] Leung Y-W 1995 *IEEE Trans. Reliab.* **44(3)** 419–427