Using Blockchain Technology to Improve N-Version Software Dependability

Denis V. Gruzenkin[1], Anton S. Mikhalev[1], Galina V. Grishina[1] and Roman Yu. Tsarev[1]

[1] Siberian Federal University, Russia
gruzenkin.denis@good-look.su
asmikhalev@yandex.ru
ggv-09@inbox.ru
tsarev.sfu@mail.ru

Abstract. Being a technique ensuring the dependability and fault tolerance of software, the N-version programming has proven its effectiveness. A formal definition and some practical experience support the idea that redundancy and diversity are the key points of the N-version software dependability. The implementation of N functionally equivalent versions allows to resist different types of faults, including residual ones. However, due to some peculiarities of N-version software design interversion and intermodule dependences can arrive. It results in the dependency of potential faults in versions or modules of the N-version software. The recently appeared blockchain technology can be applied to increase the dependability of N-version software. In the paper the authors suggest an approach to log N-version software faults by the means of the blockchain technology. As a result, the blockchain technology provides complete data on operation of the N-version software that is used to improve the N-version software dependability. An example illustrating the proposed approach is provided.

Keywords: N-version software, software dependability, blockchain, software reliability, logging.

Introduction

The information technology industry is characterized with the disappearance of out-of-date methods and the arrival of up-to-date ones. The blockchain technologies are considered one of the latest achievements in the IT [1], [2]. Nowadays they attract great attention because they have revolutionized the fields of information security and distributed data processing [3], [4].

Blockchain is a multifunctional and multilevel information technology assigned for the reliability of both tangible and intangible assets registration [5], [6]. All infor-

mation and assets operations are coded. Then they are converted into the so-called block. Any asset corresponds to both a private key and a public key. The public key is essential for the asset operations and the private key is necessary for the asset operations validity checking [7], [8]. An example of such operations is financial transactions.

The transaction is approved in case of private and public keys matching. The keys themselves are individual for any asset. They are generated according to an algorithm and look like a set of symbols. To complete the block (fix an event) a user fits the key which is connected to the previous transaction only. It results in a chain of blocks or a blockchain system. Such method eliminates the substitution of former data. This peculiarity is an issue of interest for many researches.

The illustration of blockchain functioning is the cryptocurrency Bitcoin [1], [9], [10]. A user gets a certain amount of bitcoins for a block completion. This amount is reduced with each further completion but, at the same time, a fee for a transaction increases. The users do not complete the blocks manually, but they use computer technologies instead. In other words, people direct computer power to solve difficult mathematical tasks.

However, blockchain technologies are not restricted by the financial sphere only. They are widely used in other fields of human life. There is a hypothesis about the application of blockchain technology for increasing the N-version software reliability, which is a part of dependability. The problem of software dependability is crucial in such fields as nuclear power, finance, space exploration, etc.

1 Research Hypothesis

1.1 Hypothesis Description

The concept of N-version software implies parallel or sequential execution of diversified program components (multiversions) [11]. These components should be functionally equivalent, and they should belong to the same module of the common environment [12]. The results of program components calculations are assessed, and their correctness is defined by a voting method [13]. The correct results are considered as the results of the whole module operation and the executed multiversions belonging to that module. The voting results are the more correct the more versions are implemented by the module. As multiversions are diversified, they have considerable differences in their implementation [14]. It results in the independence of their potential failures. If one of the versions gives an error, the others will return correct results. It guarantees fault tolerance of the whole software system.

Failure exposure and elimination are issues of the day in the N-version programming. The blockchain technology can be used to solve these problems. Besides, it can identify the dependence between failures of different versions and modules. We suggest applying the blockchain technology as a logging tool.

A log is a record of the system information about the operation of the N-version software system. This record is implemented in the form of files or the chains of blocks. During the N-version software run-time, all executing operations are recorded.

It allows the software programmers to identify the exact place of failure and to eliminate it by the means of proper tools.

Logs registration is not safe. The data can be changed or even removed by a programmer or a hacker. Moreover, the operations with data have risks of data loss and data distortion. Due to blockchain it is possible to log transactions without falsification and data loss. It increases the amount of logs and decreases the amount of residual errors.

Still there is a possibility for hackers to connect their equipment to the information system of an enterprise. The power of the equipment should be at least 1% higher than the power of the enterprise as a whole. In this case they can eliminate or distort logs. Nevertheless, this article is concerned with another way of information security. In this context, the possibility of correct and full logging of all transactions is equal to 100%.

1.2 The theoretical ground for the hypothesis

Errors can be both residual and obvious. They can appear at any stage of the software lifecycle. Therefore, the task is to define the log error types which are useful for finding and eliminating the errors. Special attention should be paid to residual errors as they appear only during program execution. Its appearance can be unnoticed. It means that there are no messages about errors, but some logs can indicate the correlation of multiversions or even modules operation faults.

Log-messages can be classified in different ways. It depends on the character of the task. This study is concerned with the following log classification:

1. Standard error messages provided by a multiversions design engineer – H_1;
2. Unexpected system failures messages - H_2 (generated by the programming environment);
3. Messages indicating a joint application of shared hardware by some versions implicitly - H_3;
4. Messages indicating a joint application of shared hardware by some modules implicitly – H_4;
5. Messages indicating the lack of correct data exchange between some versions and execution environment implicitly – H_5;
6. Other messages – H_6.

The sum of error *detection probabilities* of every type is equal to the following equation:

$$\sum_{i=1}^{m} P(H_i) = 1$$

where m is a number of message classes (in our case m=6).

A is taken for the detection event of a residual error in logs. The probability of this event appearance $P(A)$ is the indicator that defines the probability of increase in

multiversion software system reliability. The error elimination depends on the probability of error detection during software testing and operation. This increases N-version software dependability.

It is possible to calculate the error detection as follows:

$$P(A) = 1 - \prod_{i=1}^{m}(1 - P(H_i)) \qquad (1)$$

where event A may take place during the execution of one of the H_i-events, i. e. while the detection of one of the classified messages according to the results of a logs set analysis.

In common case the probability of errors appearance for each message class is calculated using the equation:

$$P(H_i) = 1 - (1 - p_i)^n$$

where n is the amount of messages in a log-file or a logs chain, p is the probability of error messages appearance in logs. As far as the N-version software system is a modular-based one, it is possible for versions and modules to operate simultaneously. It means that the events of logs records are independent.

2 Experiment results

The effectiveness of blockchain technology application for the residual error detection has been assessed. The value of residual error detection probability is used as an effectiveness criterion. Residual errors can be diagnosed according to the number of messages in logs. Logs analysis of the operation results of a series of multiversion software allowed to distinguish some types of independent messages. They can indicate some faults in operation (abnormal behavior):

1. the allocation of a correct total operation result of one of multiversions without intermediate data output;
2. the exceeding of a permissible limit of some multiversions operation time;
3. the increase of a random access memory capacity consumed by the module;
4. the impossibility for the N-version software environment to address to the memory containing a module operation result;
5. calculating multiversions errors on some sets of processing data.

The abundance of faults appearance in software operation is set by the probabilities:

$p_1 = 0.00004$;
$p_2 = 0.00002$;
$p_3 = 0.00003$;
$p_4 = 0.000008$;
$p_5 = 0.00006$.

The probability of a residual error detection *P(A)* can be calculated with the equation (1). Logs analysis allows to discover one of possible abnormal behavior messages in software operation. The probabilities values are shown in Table 1.

Table 1. The experiment results.

Amount of messages (n)	$P(H_1)$	$P(H_2)$	$P(H_3)$	$P(H_4)$	$P(H_5)$	$P(A)$
100	0.0039920	0.0019980	0.0029956	0.0007997	0.0059822	0.016
250	0.0099504	0.0049876	0.0074721	0.0019980	0.0148885	0.039
500	0.0198017	0.0099503	0.0148883	0.0039920	0.0295553	0.076
1000	0.0392113	0.0198015	0.0295549	0.0079681	0.0582372	0.146
5000	0.1812725	0.0951635	0.1392940	0.0392107	0.2591885	0.546

Some logs chains contain more messages about multiversion software. This allows to detect and eliminate a residual error with higher probability. It is vital at an early testing stage.

Conclusion

The logging of information on multiversions execution along with extra multiversions intended to improve the software dependability increase a time period for software design and development. Besides, there is a negative effect of the blockchain technology, such as a low speed of transactions.

However, the blockchain technology application allows both to detect errors and to identify interversion and intermodule dependences at the stages of software testing and operation. The more multiversions provide a correct result the more dependable the N-version software is. Finding an exact problem source during the multiversions execution allows not only to debug and improve the program code, but also to prevent a program component from failure when the whole multiversions set is implemented.

References

1. Beck, R.: Beyond Bitcoin: The Rise of Blockchain World. Computer 51 (2), 54-58 (2018).
2. Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C., Santamaria, V.: To Blockchain or Not to Blockchain: That Is the Question. IT Professional 20(2), 62-74 (2018).
3. Hawlitschek, F., Notheisen, B., Teubner, T.: The limits of trust-free systems: a literature review on blockchain technology and trust in the sharing economy. Electronic Commerce Research and Applications 29, 50-63 (2018).
4. Khan, M.A., Salah, K.: IoT security: review, blockchain solutions, and open challenges. Future Generation Computer Systems 82, 395-411 (2018).
5. Hughes, T.M.: The global financial services industry and the blockchain. Journal of Structured Finance 23 (4), 36-40 (2018).

6. Xu, C., Wang, K., Guo, M.: Intelligent resource management in blockchain-based cloud datacenters. IEEE Cloud Computing 4(6), 50-59 (2018).
7. Chen, Z., Zhu, Y.: Personal Archive Service System using Blockchain Technology: Case Study, Promising and Challenging. In: Proceedings - 2017 IEEE 6th International Conference on AI and Mobile Services, AIMS 2017, pp. 93-99. IEEE (2017).
8. Drescher, D.: Blockchain Basics: A Non-Technical Introduction in 25 Steps. Apress (2017).
9. Hong, K.H.: Bitcoin as an alternative investment vehicle. Information Technology and Management 18 (4), 265-275 (2017).
10. Kaushal, P.K., Bagga, A., Sobti, R.: Evolution of bitcoin and security risk in bitcoin wallets. In: 2017 International Conference on Computer, Communications and Electronics, COMPTELIX 2017, pp. 172-177. IEEE (2017).
11. Avizienis, A., Chen, L.: On the implementation of N-version programming for software fault-tolerance during program execution. In: Proc. IEEE Comput Soc Int Comput Software & Appl Conf, COMPSAC, pp. 149-155 (1977).
12. Gruzenkin, D. V., Chernigovskiy, A. S., Tsarev, R. Y.: N-version Software Module Requirements to Grant the Software Execution Fault-Tolerance. Advances in Intelligent Systems and Computing. vol. 661, pp. 293-303. Springer, Heidelberg (2017).
13. Durmuş, M.S., Eriş, O., Yildirim, U., Söylemez, M.T.: A new bitwise voting strategy for safety-critical systems with binary decisions. Turkish Journal of Electrical Engineering and Computer Sciences 23 (5), 1507-1521 (2015).
14. Baudry, B., Monperrus, M.: The multiple facets of software diversity: Recent developments in year 2000 and beyond. ACM Computing Surveys (CSUR) 48(1), 16 (2015).