ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter42892020.pdf
https://doi.org/10.30534/ijeter/2020/42892020

Performance Comparison of Retry and N-Copy Software Fault Tolerance Techniques

Mylara Reddy C.[1*], Y. Vamsidhar[2], Mohan Gowda V.[3], B. Ramesh Naik[4]
[1]Department of CS,. GITAM University, India, mchinnai@gitam.edu
[2] Department of CSE, GITAM University, India, vyendapa@gitam.edu
[3] Department of CSE, GITAM University, India, mgowda@gitam.edu
4 Department of CSE, GITAM University, India, rameshnaik.bhukya@gitam.edu

ABSTRACT

Day by day every individual and small to large organizations are moving towards the use of cloud computing systems for their daily activities. Cloud computing paradigm provides different service models such as Platform-as-a-Service, Software-as-a-Service, Infrastructure-as-a-Service, and etc. Platform-as-a-Service enables users to host their software systems. It's important to ensure that the software systems hosted on cloud environment provide reliable service to users. There are different fault tolerance techniques which help software engineering to prevent software systems failure. In this research article an attempt is made to compare performance of the following software fault tolerance techniques: 1) Retry Block (RtB) and 2) N-copy programming. The performance of these approaches was measured in time taken to recover from failures which are introduced into Fibonacci numbers at random to impersonate failures. Based on the results of experiments conducted on Amazon cloud using the Merit Trac's LMS portal, it is found that RtB approach is the most optimal when retry blocks are smaller in size compared to N-Copy. In case of few failures Retry is better than N-Copy and N-Copy perform better than Retry block technique in case of frequent failures.

Key words: Reliability, fault tolerance, Retry-block, N-Copy

1. INTRODUCTION

Due to the increased complexity and size of software systems, the reliability of software systems has recently become a looming and unsolved problem in any computing environment for example, cluster computing, cloud computing, and high performance computing. There are several software fault tolerant methods proposed by researchers which are broadly categorized based on criteria such as reactive, proactive, design diversity, data diversity, static, dynamic, and hybrid fault tolerance techniques. Further, under each category, there exist several approaches contributed by the research community since 1980's. The cloud computing paradigm has become the most prominently used technology due to it's easy to access anywhere at any time and cost-benefit. As a result of it, private clouds are expected to host large and complex proprietary software systems and software development platforms for large business organizations to meet their customer requirements. As the number of users at a given instant of time increase, cloud based software systems should be scalable well to meet the demand during peak time. With increased number of users and increased complexity of software systems, there are higher chances of occurrence of hardware and software failures.

There are several reasons which attribute to the failure of software systems or software applications that are designed and developed for distributed computing environment, particularly cloud computing environment. In [1] O. Gadish list the following top Nine reasons for cloud application failure: 1) Operator or Human Errors, 2) Application Bugs, 3) Cloud Provider Downtime, 4) Extreme Dynamics in Customer Demand, 5) Quality of Service, 6) Third Party Service Failures, 7) Security Breaches, 8) Hardware Failures, and 9) Lack of Disaster Recovery Plan. In addition to the above nine reasons, real time applications fail due to time out. Traditional distributed software fault tolerant systems were able to successfully thwart the effects of software system failures by the execution of static and dynamic fault tolerance techniques such as redundancy, checkpoint restart, migration, software rejuvenation, self checking techniques on extra/additional computing resources. However, due to decrease of mean time between failures (MTBF) of distributed computing systems like cloud computing technology, grid computing, and cluster computing traditional fault tolerant techniques for software resilience either do not scale to meet systems increased demand for performance or require too much of resources like hardware, energy, and time to be feasibly implemented.

Several software fault tolerance techniques that are contributed by researchers are based on design diversity and data diversity. Many of these techniques in recent works were able to handle failures associated with cloud based software systems [2]. However, their relative performance, reliability

and fault tolerance capability and scale of resource requirements of fault tolerant techniques, particularly techniques which work based data diversity, when compared with one another is unknown or unclear.

Each fault tolerant technique's performance is simulated with varying system sizes as well as varying degrees of severity due to software system failures. Strengths and weakness of each fault tolerant technique is analyzed and compared.

In this work we make the following contributions:

- We design and develop a model for simulating the each technique in terms of time taken to complete the task of recovering from failures, particularly in a cloud computing environment
- We provide a relative comparison of the performance of software fault tolerance techniques in cloud computing environment.

2. RELATED WORK

Although there are several other techniques for mitigating the effects of system failures, in this article we work on software fault tolerance techniques that are considered to be transparent to software application programmers and users of software systems. We refer the interested reader to the summaries provided for all such techniques in [3, 4, 5, 6, 7, 8, 9]. Device layer or sensing layer in IoT networks is one of the critical places at which faults may occur. Handling failures/faults in this layer is very important. Satry *et al.* [12] have proposed fault tolerant approaches using sub-nets topologies to determine level of fault tolerance of IoT networks. Ramesh B. *et* al. have presented an approach to improve the reliability of object detection and classification using image processing[13]

2.1 Retry Blocks (RtB)

The RtB technique is one of the two original data diverse software fault tolerance techniques proposed by Ammann and Knight [10]. It is the data diverse complement of design diversity and it is categorized as dynamic technique. It makes use of acceptance test and backward recovery approach to achieve fault tolerance. Retry technique consists of following components to accomplish fault tolerance: 1) Primary Algorithm, 2) Data Repression Algorithm (DRA) 3) A Watchdog Timer (AWT), 4) Backup algorithm, and 5) Acceptance Tests (AT). Figure 1 shows the structure of a retry block. Figure 1 shows the pictorial representation of retry block technique.

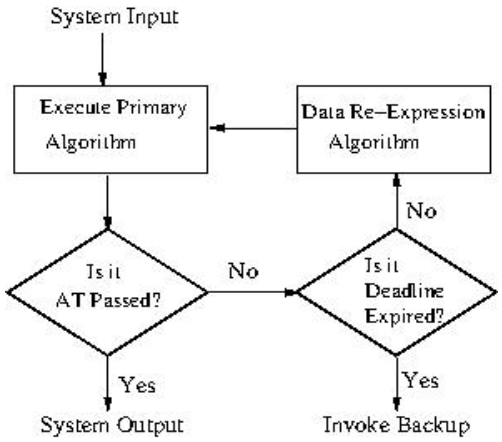


Figure 1: Retry block technique

In RtB approach primary algorithm executes and evaluates its output at the end of every execution using acceptance test. If the acceptance test passes, then the retry block is complete. In case of acceptance test failure, primary algorithm executes again using re-expressed data as input. This process continues until specified number of attempts exhausted, thereby invoking the backup algorithm or produces a valid output.

2.2 N-Copy Programming (NCP)

N-copy programming, as shown in figure 2, is a data diversity complement of N-version programming that is design diversity based technique. Each copy of N copies of a program executes in parallel on a data set produced by data re-expression algorithm. The output of the N-copy programming is selected by using voting mechanism. Figure 2
shows the structure of the N-copy programming system.

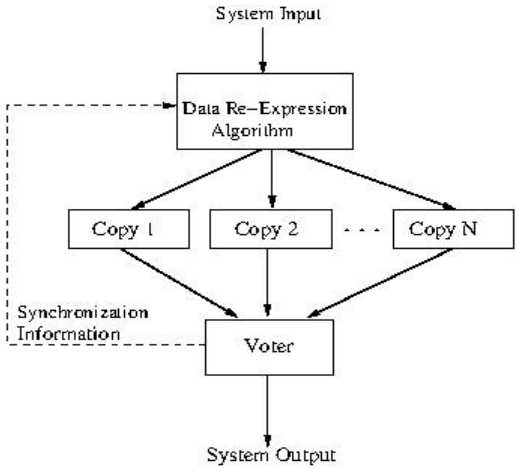


Figure 2: N-Copy Programming

To determine the performance of the N-copy system, we have considered three-copies of a program and compare it with a single version system. Inputs of N copies can map to A + UA

different outputs. A of the N outputs are accepted by the system and the probability that the i^{th} copy produces acceptable output is a_i for i=1...A. Similarly F of N outputs are not acceptable by the system and the probability that i^{th} copy produces unacceptable output is f_i for i=1...F

Three copies are run separately using one of three data sets as an input. These data sets are generated by the data re-expression algorithm. The voter selects the output that has occurred most frequently. In case of a tie in the output, the tie is resolved in random fashion. Given the probabilities ri and fi, the system will select an acceptable output in one of the four possible choices.

1. All the three copies of a program maps all three re-expressed data sets to acceptable outputs. This is achieved with probability expressed using the equation 1.

$$P1 = \sum_{i=1}^{A}\sum_{j=1}^{A}\sum_{k=1}^{A} ai * aj * ak \qquad (1)$$

2. Of all the three copies of a program, two copies mapped to two of the three re-expressed data sets to a same acceptable output and one data set to an unacceptable output. It is possible with probability expressed using the equation 2.

$$P2 = \sum_{i=1}^{A}\sum_{j=1}^{F} ai * fj \qquad (2)$$

3. Of all the three copies of a program, two copies mapped to two of the three re-expressed data sets to different acceptable outputs and one data set to an unacceptable output. It is possible with the probability expressed by using the equation 3.

$$P3 = \sum_{i=1}^{A}\sum_{j=1}^{A}\sum_{k=1}^{F} ai * aj * fk \qquad (3)$$

such that $i \neq j$.

4. Of all the three copies of a program one copy map one of the three re-expressed data sets to acceptable output and two data sets to an unacceptable output. It is possible with the probability expressed by using the equation 4.

$$P4 = \sum_{i=1}^{A}\sum_{j=1}^{F}\sum_{k=1}^{F} ai * fj * fk \qquad (4)$$

such that $j \neq k$.

The probability that a three copy system will produce an acceptable output is expressed by using the equation

$$P_{acceptableoutput} = P1+P2+P3+P4$$

2.3 Data Re-expression Algorithm (DRA)

Data re-expression is the process of transforming input data into logically equivalent data sets. Figure 4 shows the basic structure of data re-expression. In simplest form, an input x given to a re-expression algorithm, say P, it generates an output P(x). Data re-expression algorithm (DRA) transforms the input x into output y=DRA(x). The original input x and transformed input y are equivalent, but approximates of each other. Using data diversity, output, P(y), of transformed input y may tolerate faults when its output is suitable but output P(x) of x is not suitable. Requirements of data re-expression algorithms are driven by the output of applications or an execute since outputs are very important. Therefore, requirements for data re-expression algorithms are gathered from the output of a given application.

3. OVERVIEW OF THE PROPOSED WORK

Although there are several techniques to mitigate effects of system failures, in this article we work on software fault tolerant techniques that are considered to be transparent to software application programmers and users of software systems. In this work we measure the time complexity of a program that generate the Fibonacci numbers and count the number of Fibonacci numbers within the given input range.

3.1 Software Failures Model

An unsigned integer is divided into four groups of one byte each. Further severity of an error quantified based on the position of the byte in which an error is occurred. In this work, an error or a fault is induced into the system by flipping the value of a bit. Selection of a particular group and one or more of its bits to induce an error is done in a random manner. The probability of failures occurring in a software system is modeled according to the exponential distribution [11]. Failures are classified into following four categories:

i. **Ignorable Failures:** Presence of errors in group 1 are ignored since the chances of mismatching program output with correct output is less.

ii. **Minor Level Failures:** Change in the value of bit(s) that belong to the second group (second least significant byte). Minor faults will be corrected by re-iterating few iterations of the Fibonacci number generation process and appropriately resetting the count of Fibonacci numbers

iii. **Medium Level Failures:** Change in the value of bit(s) that belong to group 3 considered as medium level failures. Medium level failures will be corrected by beginning execution from the previous checkpoint.

iv. **High Level Failures:** Change in the value of bits(s) of most-significant byte (fourth byte) considered as high level failures. These failures are eliminated by reexecuton of Fibonacci number generation process from the beginning

4. EXPERIMENTAL SETUP AND RESULTS DISCUSSION

In this section we present the details of implementation of proposed work in detail and discussion of the results in detail.

4.1 Retry Blocks

To measure the performance of RtB approach, we have programmed proposed algorithm using C++ programming language that generates the Fibonacci numbers and their count between zero and given input value. During the execution of the program, errors are introduced randomly into any one or more than one number that appear in the Fibonacci sequence by modifying values of one or more bits. As a result of the modification, the count of Fibonacci numbers will either increase or decrease depending on whether the error has resulted in reduced or increased value.

i. For example, for input value 16, the program executes with transformed first input data set (2, 4) producing the count of Fibonacci numbers. At the end of first execution the count of Fibonacci numbers is passed to the acceptance test. The acceptance test compares the program output with the actual count that supposed to be. In case of a successful match, the program does not go to retry otherwise it will retry with the second data set (4, 2).
ii. Again during the execution of the program for the second time, errors are randomly introduced into either the base value two or power value four or both. Then the output of second iteration passed to acceptance test. Depending on the decision result, the acceptance test program either stops execution or continue with the next data set (16, 1). This process continues for three different data sets.
iii. After the third attempt program return exception failure.

We have used specification acceptance test to test the retry block approach. The acceptance test determines whether the program has produced exactly equal number of Fibonacci numbers as that of expected or not. If the output matches expected value, then the acceptance test is considered as passed otherwise failed.

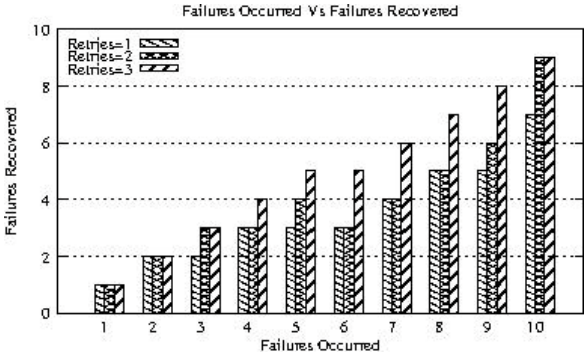
4.2 N-Copy Programming

In this approach, three copies of a C++ program to generate and count Fibonacci numbers between zero and given input were used. Input for each copy is selected from the transformed input data sets. For example, data sets (16, 1), (2, 4), (4, 2) are the input for the first, second and third copies respectively. The outputs of each copy, that is the count of Fibonacci numbers, are passed to voting mechanism. The voting mechanism is a straight forward approach that selects identical output.

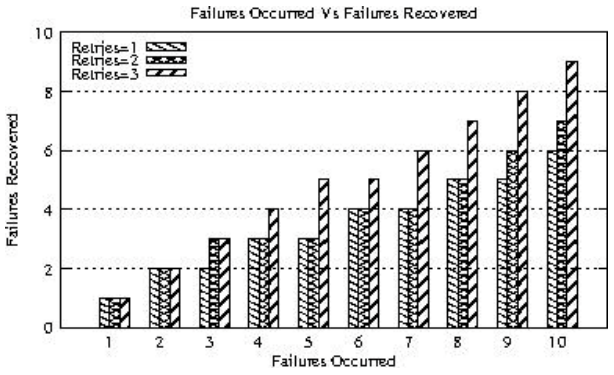
As the number of retries increases time taken by each subsequent retry attempt also increase. There are four levels of faults that are dealt with in our experimental programs. Minor level failures are basically introduced into the second least significant byte of an input number. Minor level failures do not introduce a much difference in value compare to higher level failures. These failures are recovered by reiterating through a maximum of two immediate previous iterations. Reason for two previous iterations is that the count of Fibonacci numbers for a smaller range will be less. Next level failures are medium level failures which are introduced in to the third byte from least significant position. Since medium level failures cause reasonably higher difference in value, it requires little more computing power to recover from failures. To annihilate the effect of such errors reiterate through five immediate previous iterations. For higher level failures entire process of counting Fibonacci numbers is repeated from the beginning of a program.

Figures 3, 4 and 5 showcases the number of failures tolerated for minor, medium and high level failures respectively using retry block approach. A higher number of minor failures can be recovered using few retries. The same is not true with higher level failures. However, a large number of failures can be tolerated with increased number of retries. Results show that with two or three retries it is possible to tolerate all minor failures and most of higher level failures.

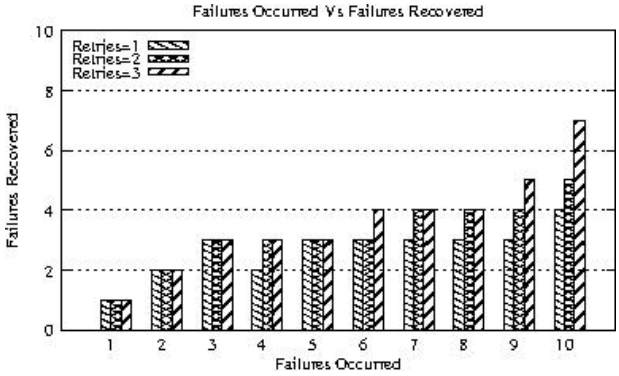
Outputs of three copies are submitted to the voter that selects the correct output using a specification test. The voter selects an output that is identical for two of the three copies. For each given input all the three copies are executed for all possible failures. Since all the copies are executed in parallel, time taken by this approach is the sum of the time taken by each copy. Time taken for each input and for each failure level varies depending on the count of Fibonacci numbers in a given range. The larger input range will have more number of Fibonacci numbers. As a result of it the time taken also increases.



Figures 3: Number of minor level failures recovered during 1, 2, and 3 retries



Figures 4: Number of medium level failures recovered during 1, 2, and 3 retries



Figures 5: Number of high level failures recovered during 1, 2, and 3 retries

Time taken by the NCP is almost thrice the original program. However, it varies depending on the computing platform on which copies are executed. Figures 6, 7 and 8 represent the number of minor, medium and high level failures tolerated by the system using NCP approach. As we increase the number of copies, the number of failures tolerated by the system also increases particularly minor and medium failures. Results show that by increasing the number of copies it is possible to tolerate most of the failures.

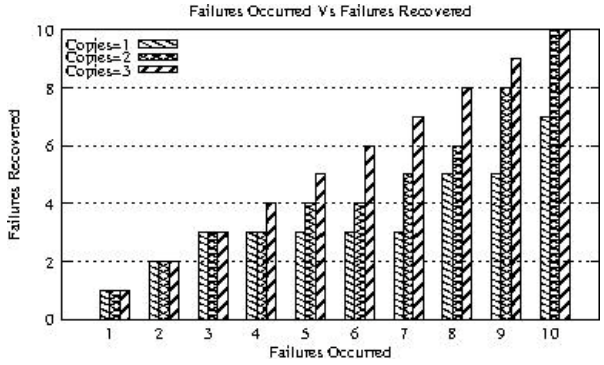


Figure 6: Number of minor level failures recovered for 1, 2, and 3 copies

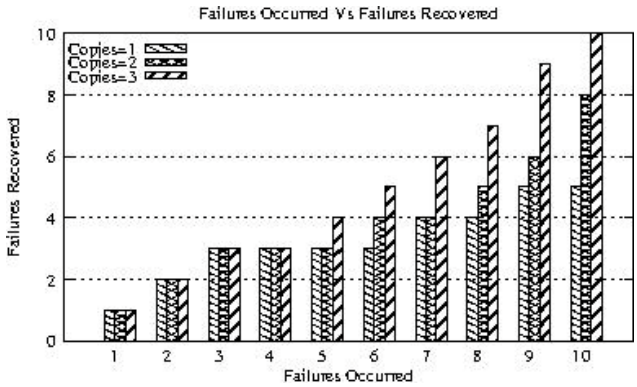


Figure 7: Number of medium level failures recovered for 1, 2, and 3 copies

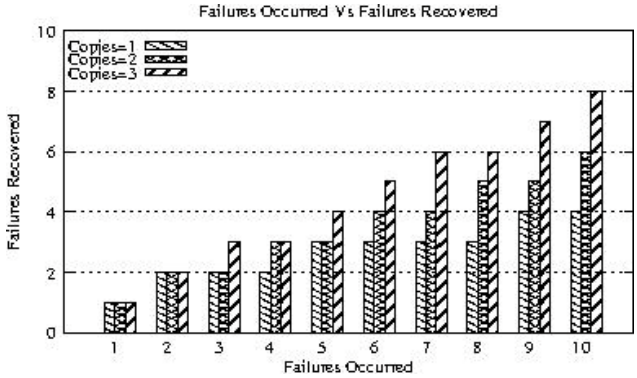


Figure 8: Number of high level failures recovered for 1, 2, and 3 copies

5. CONCLUSION

In this research article we presented performance comparison of retry block and N-copy techniques. Experimental results show that the N-copy is more efficient than the retry block approach in terms of the number of failures tolerated. However, in terms of time taken to complete a task retry block approach is efficient when the size of the retry blocks is smaller. Efficiency of N-copy programming depends on N. Higher the value of N then the larger the amount of resources required.

REFERENCES

1. Ofer Gadish **Top 9 Reasons for Cloud Application Failure,** (2014), https://www.cloudendure.com/blog/top-9-reasons-cloud-application-failure/
2. Ganesh, A., Sandhya, M., Shankar, S. **A study on fault tolerance methods in cloud computing,** *IEEE International Advance Computing Conference (IACC).* pp. 844–849 (Feb 2014)
3. Egwutuoha, I.P., Levy, D., Selic, B., Chen, S. **A**

survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems, *Journal of Super computing*, 65(3), 1302–1326, http://dx.doi.org/10.1007/s11227-013-0884-0

4. Elnozahy, E.N.M., Alvisi, L. Wang, Y.M., Johnson, D.B. **A survey of rollback recovery protocols in message passing systems**, *ACM Computing. Survey*, 34(3), 375-12 408 (Sep 2002), http://doi.acm.org/10.1145/568522.568525
5. Fagg, G.E., Dongarra, J. **Ft-mpi: Fault tolerant mpi, supporting dynamic applications in a dynamic world**, *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. pp. 346–353, Springer-Verlag, London, UK (2000), http://dl.acm.org/citation.cfm?id=648137.746632
6. Sharma, P., Phanden, R. K., & Singhal, S. A Comparative Analysis of Facility Layout Design and Optimization Techniques.
7. J. F. Ruscio, M.A.H., Varadarajan, S. **Dejavu: Transparent user-level checkpointing, migration, and recovery for distributed systems**, *IEEE International Parallel and Distributed Processing Symposium*, pp. 1–10 (March 2007)
8. Limam, S., Belalem, G. **A migration approach for fault tolerance in cloud computing**, *International journal of. Grid High Perform. Computing*, 6(2), pp. 24–37 (Apr 2014), http://dx.doi.org/10.4018/ijghpc.2014040102
9. Maloney, A., Goscinski, A. **A survey and review of the current state of rollback-recovery for cluster systems**, *Concurrency and Computation Practice and Experience* 21(12), pp.1632–1666, http://onlinelibrary.wiley.com/doi/10.1002/cpe.1413/pdf
10. S., K., V, R. **A survey of checkpointing algorithms for parallel and distributed computers**, *Sadhana* 25(5), 489–510 (Oct 2000), https://doi.org/10.1007/BF02703630
11. Ammann, P.E., Knight, J. C. **Data diversity: an approach to software fault tolerance**, *IEEE Transactions on Computers* 37(4), 418–425 (Apr 1988)
12. JKR Sastry, Bhupathi, Enhancing **Fault Tolerance of IoT Networks within Device Layer**, *International Journal of Emerging Trends in Engineering Research*, 8(2), February 2020, 491-509
13. Bhukya Ramesh Naik, Vamsidhar Yendapalli, Naga Raju M., **Performance Improvement in CBIR using Region Weight Learning Approach,** *International Journal of Emerging Trends in Engineering Research*, 8(7), July 2020, 3864-3868