Fault Tolerance Techniques and Applications for Distributed Systems

S.K. Shehan
Faculty of Computing
Sri Lanka Institute of Information Technology
it20206864@my.sliit.lk

B.K.R. Dilshan
Faculty of Computing
Sri Lanka Institute of Information Technology
it19967080@my.sliit.lk

H.A.S. Jayasinghe
Faculty of Computing
Sri Lanka Institute of Information Technology
it20222222@my.sliit.lk

T.P.D.Pieris
Faculty of Computing
Sri Lanka Institute of Information Technology
it20272586@my.sliit.lk

***Abstract*—In the past people used centralized systems mostly. But because of gradual development in network and design technology distributed systems were introduced. Distributed systems organize components to hide the underline existence from the user. Therefore it is very challenging to maintain several components. Fault tolerance is one of the major challenges that designers should consider. Replication, checkpointing, n-version programming, Byzantine Fault Tolerance are the techniques discussed in this research paper by referring reviews published after 2017. We discuss how each technique relates with fault tolerance in detail. In addition, this review highlights key challenges and modern solutions for them.**

Keywords—: Fault tolerance, Replication, Checkpointing, N-version programming, Byzantine Fault Tolerance

I. Introduction

Distributed system is considered as a group of components which manipulate the same kind of task [1]. In distributed systems there can be three kinds of issues. They are faults, errors, and failures. These are coupled with each. Fault is the core issue of the problem. Result of fault is known as error and failure refers to the last outcome. According to the report provided by [2] there can be different types of faults. Omission faults, aging related faults, response faults, software faults, timing faults and miscellaneous faults might have occurred in the system. If a distributed system needs to become less faulty it should close to a dependable system. Dependable system has its key features. System is able to run 24 hours without any failures. Whenever some components are damaged other components should not affect other components. More importantly, the system must execute actions for sudden failures without noticing users of the system. Fault tolerance enables the system to continue although the system encounters faults that might affect efficiency. The prime objective of this review paper is to discuss fault tolerance techniques in order to reduce risks that can occur in distributed systems. Therefore, initially faults should be detected and require recovery. [1] Main goal of these techniques is to detect faults and provide solutions as soon as possible.

II. Background

In this review, four main fault tolerance techniques in distributed systems are discussed by using previous studies published after 2017 which are arranged under the reference section. Although there are so many fault tolerance techniques, we sort out the techniques that have been implemented until now. Throughout the discussion we discuss each technique in detail. We explore the background and key features of technique. Our main focus is to investigate how the technique is related to the distributed system. We aim to convey Challenges and solutions for technique as well. In addition much attention has been drawn to express our opinion in contrast with referenced materials.

III. DISCUSSION

A. Replication Based Method for Fault Tolerance

The replication-based method in fault tolerance falls under the reactive fault tolerance category , hence by this method , faults will be managed after they occur. This is considered to be one of the most popular fault tolerant techniques. In simple terms , what this method does is replicate the data to different other systems , so that when a fault occurs in one system , the other system would continue running. But there is a problem in this , more such backup systems can cause redundancy , thus it can be considered as a trade-off between consistency and efficiency , because even though consistency is achieved , it will become less efficient because of redundancy which can cause many problems which will be discussed ahead. This replication is managed as in the figure below. [3]

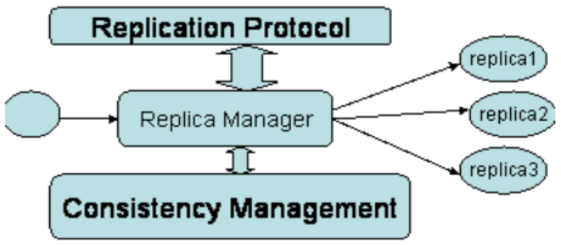

Figure 1: Replica Management

According to the replication protocol , the components in the replicas will get affected and the consistency management will help in maintaining up to date components in the replicas as for the main system.

Although the Replication technique is considered to be one of the most famous methods in fault tolerance , there are few major issues like consistency , degree of replica , replica on demand.

Consistency Issue

Due to updates on an entity , their copies also have to be updated accordingly and this causes consistency issues. Consistency is assured by the replication protocol which has many types of criterion also which can be categorized into strong consistency criterion and weak consistency criterion. linearizability(Linearizability is one of the strongest single-object consistency models, and implies that every operation appears to take place atomically, in some order, consistent with the real-time ordering of those operations: e.g., if operation A completes before operation B begins, then B should logically take effect after A.) and sequential consistency(Sequential consistency is a strong safety property for concurrent systems. Informally, sequential consistency implies that operations appear to take place in some total order, and that that order is consistent with the order of operations on each individual process) define strong consistency criterion, whereas causal consistency defines a weak consistency criterion. For replication , mainly there are two strategies ; Passive strategy and active strategy. In passive strategy , only the primary executes requests and multicasts state changes for other replicas. But in active strategy , the request is sent to all replicas and gets executed by all of them , thus the active strategy uses lower network resources and it also responds to faults faster than in passive strategy.

Degree of Replica

Degree of replica means the number of replicas. This may give low performance both by having a high number and a lower number , high number of replicas will lead to higher maintenance cost in the other hand less number of the replicas would affect the performance , so it is required to have a reasonable amount of replicas , also , this number depends on system configuration and load. Researchers have proposed algorithms to create such an adaptive number of replicas.[4]The replica creation based on access tendency (DRC-AT), the replica placement based on user request response time and storage capacity (DRP-RS) and the replica selection based on response time (DRS-RT) are proposed. The DRC-AT algorithm introduces the two parameters of file popularity and period value of file popularity, calculates the file access tendency periodically and decides the creation and deletion of the replica of the file according to the size of the file access tendency. The DRP-RS algorithm evaluates the user's request response time and storage capacity to select the best node set to place the replica. The DRS-RT algorithm returns to the user the node with the strongest service capability that contains the user's requested data. Experiments show that the algorithm can improve the speed of data reading by the client, improve the resource utilization, balance the load of the node and improve the overall performance of the system

Applications

Fault Tolerant Containers using NiLiCon [5]

In most of the scenarios , distributed systems rely on virtual machines or containers for replication. In NiLiCon , these two have been combined for maximum efficiency. Thus it uses both virtual machines(VM) and containers , known as Remus and CRIU respectively. It is pointed out that Remus is a passive replication strategy , but as pointed in [3] , active strategy is more effective than passive strategy , however this is how NiLiCon has been built.

Fault Tolerant Method for Network Distributed Flight Control System based on Task Replication

The aeronautical field requires a very high level of fault tolerance as it carries a lot of risk. This proposed design is to ensure better fault tolerance in the system. [6] Scheduling scheme based on task replication is the key technology to improve fault tolerance of distributed systems. In a single processor node system, through the system task to reproduce, they can be identified as primary tasks and backup tasks. Of course, a primary task can have multiple backups, therefore, even if there is damage in the primary task, the task will be transferred to the backup task, the whole system execution still won't be affected, in the other words, it has a very good fault tolerance. In the distributed system with several computer nodes, the tasks also divide into the primary tasks and backup tasks, but they will run on a different computer node respectively, can ensure that a particular computer nodes failure, redundant tasks on another computer node can still perform before the deadline, so as to achieve the goal of fault-tolerant. Therefore, the scheduling method based on task replication is to realize fault tolerance through task redundancy, which is the best choice in the application of reliability systems. However, the influence of introducing task redundancy on the system's schedulability and resource utilization should also be considered.

However , in the above described solution , it seems that the degree of replica which was discussed in [4] is not taken into consideration.

Replication-Based Fault-Tolerance for Large-Scale Graph Processing

Large-Scale Graphs are mostly used to express many machine learning and data mining (MLDM) algorithms. Here , computations are coded in vertices and the communication between them happens through edges , this is the basic idea of a graph algorithm. With the increase of the complexity of such algorithms , stable fault tolerance techniques have been required.In such systems , fault tolerant methods like checkpointing or replication are used. In checkpointing , whenever a fault occurs , it restores the most recent non-fault state.But this caused lengthy recovery times which have been notable because of reloading from slow persistent storage.This research has pointed out a new method “Imitator” which is based on the replication-technique. The “Imitator” has extended the present fault tolerance method in three ways.
First, Imitator extends the existing graph loading phase with fault tolerance support, by explicitly creating additional replicas for vertices without replication. Second, Imitator maintains the freshness of replicas by synchronizing the *full states* of a master vertex to its replicas through extending normal messages. Third, Imitator extends the graph-computation engine with fast detection of machine failures through monitoring vertex states and seamlessly recovers the crashed tasks from replicas on multiple machines in a parallel way, inspired by the RAMCloud approach.[5]
Thus , “Imitator” , has been introduced to overcome the issues of the previous fault tolerance techniques which have been used.

B. N-Version Programming

N-version programming is a fault tolerance technique used in distributed systems where N >= 2 functionally equivalent software modules are generated independently. These modules are called ‘versions’. These independent versions are developed by N number of individuals or groups that have no interaction with each other with respect to the development process. Different algorithms, programming languages, environments, translators and tools are used in each effort when it is possible. Usually, all the generated versions are run simultaneously, and the one that is most tolerant to faults is selected. The goal of N-version programming technique is to minimize the probability of similar errors [8].

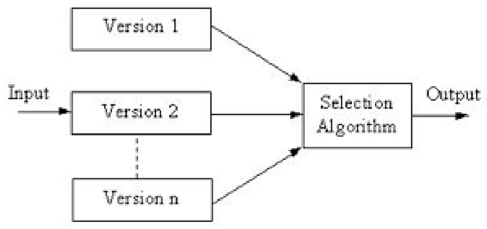

Figure 2: N-Version programming model

Before creating individual versions, an initial specification should be created. This initial specification states the functional requirements completely and unambiguously and it leaves the widest possible choice of implementation to the independent programming efforts. The specification also states all the special features that are required in order to execute the set of individually generated versions in a fault-tolerant manner. The initial specification should define: 1) the function to be implemented by the individual version; 2) cross-check points (“cc-points”); 3) content and format of the cross-check vectors ("cc-vectors"); 4) the comparison (matching or voting) algorithm; 5) the response to the possible outcomes of matching or voting.

Independent versions are developed independently according to the initial specification. Even though the implementation of these versions are different from each other, they are assumed to be functionally equivalent. Together, all these individually generated versions form an N-version software unit. However, each and every individual version must go through an acceptance testing before integrating them into the N-version software unit. To execute an N-version software unit and to match or vote the results generated by each version there are three types of special mechanisms that are used. The three mechanisms are: 1) comparison vectors (data structures that represent a subset of a version's local program state); 2) comparison status indicators (used to indicate actions to be taken after matching or voting); 3) synchronization mechanisms (used to synchronize the execution steps of an N-version software unit).

N-Version Programming in Distributed Systems

People have done research on applying the N-version programming technique to tolerate faults in distributed systems. However, the effectiveness of some of these are yet to be measured.

1. Deep Neural Networks

   Applying deep learning algorithms in modern applications has become a trend as it helps organizations to improve their systems or products. However, the reliability of deep neural networks is a problem. Deep neural networks are unreliable because: 1) they are vulnerable to data perturbations; 2) some erroneous cases are challenging to fix even though they were known beforehand [9]. To overcome this reliability issue in deep neural networks the N-version programming technique can be used.

   However, N-version programming can not be applied to deep neural networks directly. This is because deep learning models automatically learn from data instead of explicitly being coded by programmers. This makes the generation of independent versions more challenging. However, a group of researchers [9] have conducted research on applying N-Version programming to DNN. In their approach, namely NV-DNN, they have proposed three independent factors that can be introduced in the development process to generate multiple independent versions. These factors are independent training, independent network, and independent data [9].

   In the proposed NV-DNN approach, for the decision procedure, they suggest using a simple decision procedure such as voting-based. Experiments done on the NV-DNN [9] approach have revealed that it is an effective way to develop fault-tolerant DNNs.

2. Fog-based IoT Systems

   Fog-based IoT systems are distributed systems in which a series of nodes receives data from IoT devices in real time. To achieve fault-tolerance in fog-based IoT systems, N-Version anomaly-based Fault Detection (NvABFD) technique can be used as suggested by [10]. This technique helps to identify data anomalies, errors, faults and failures in fog-based environments in real time. In N-version programming technique, multiple models with similar functionalities coexist within the same system or environment. This is helpful in fog-based systems that contain sensors with similar functionalities. When using N-version programming in fog-based systems, sensors can be treated as independent versions with similar specifications. This way, it is possible to identify commonalities among errors, faults, and failures that exist in fog environments.

Advantages

- In N-version programming we follow the design diversity technique. The independently generated versions will fail independently and the probability of coincidental failure is low. This way, the N-version programming technique ensures that at least one of the independent versions will continue to provide the required functionality [8].
- In N-version programming, verification and validation of independent versions is done concurrently. This reduces the time that will be spent on software verification and validation.
- When a formal and an effective initial specification is given, different versions of the system can be developed by individuals at their time and location using their own personal computing equipment. This helps to reduce the cost of development drastically in highly controlled professional development environments [8].
- N-version programming ensures that independent versions that are implemented differently will cause very few similar errors at decision points.

Challenges

- The key factor for success of N-version programming is the accurate initial specification. If the initial specification is not clear, the N-version programming technique will not be that effective.
- Developing multiple versions of the same system with different languages, tools and environments can increase the cost of development. However, some researchers suggest that applying design diversity only to critical paths can help to reduce the cost of development.
- N-version programming can increase the software complexity. If the software complexity is high, the probability of error will also be high.

C. *Byzantine Fault Tolerance*

C.1 Byzantine Failures

Byzantine failures are the failures which do not lead to 'fail-stop' behavior, instead of that it will send conflicting messages/information which leads to incorrect results in the system without a failure such as send an incorrect response, send an intentionally misleading result as a response and send responses with different information to other parts of the system. Also known as the arbitrary failures.

This is happening due to having single or multiple malicious nodes which are hacked or owned by the hackers or sometimes due to hardware/software failures. Because sometimes, servers can be given unusual responses due to some of the errors such as not having enough resources, not handling exceptions correctly and likewise.

This is a very popular problem with the distributed systems in computing industry, and introduced as 'Byzantine Generals Problem', which is an extended version of the two-army generals problem [11], where having more than 02 nodes to communicate with. In this kind of a situation, the main goal is to make a consensus among the nodes on whether an action should be taken or not, depending on all the responses of all the nodes.

C.2 The Byzantine Generals Problem

Let's see what is this byzantine generals problem first; Several armies surround an enemy city, and each army has a general to lead their army. And also they need to attack the enemy city together at the same time by establishing a consensus between generals. Therefore, each of the generals need all the other general's responses to decide. However, some can be traitors (malicious nodes) who are deliberately sending false information in order to get the wrong decision which will lead to destroying the armies.[11] The main facts are ;

- All the generals have to decide, whether attack or retreat
- Once they established a consensus, it cannot be changed
- Messages can be get lost,delayed or destroyed due to many reasons
- One or more generals can be traitors

Considering the above facts, it means we can establish a consensus if there are at least ⅔ reliable nodes. This problem was unsolvable for years, but however many computer science specialists came up with different kinds of approaches and techniques to tolerate the byzantine faults in distributed systems to a certain extent considering unique areas. Because in most of the approaches, it will be hard to make the correct decision, if a system consists of many malicious nodes. But a solution for this problem is a must for blockchains, therefore Satoshi Nakamoto solved this problem using the Proof-of-Work(PoW) mechanism for blockchains.[19]

C.3 Byzantine Fault Tolerance (BFT)

Byzantine fault tolerance is the ability of a distributed system to tolerate the byzantine faults and reach a consensus even if some of the nodes are malicious.

There are many BFT methods and algorithms introduced considering on certain aspects such as Algorithm Oral Message(OM), Signed Message Algorithm (SM)[11] , Practical Byzantine fault tolerance (PBFT)[12], Threshold Signature Practical Byzantine Fault Tolerant(TS-PBFT) [13] (Cheng 2021), multi-block consensus algorithm [4], Credit Practical Byzantine Fault Tolerance (CBPFT), Byzantine consensus algorithm based on Gossip protocol (GBC), Vote algorithm (vBFT) [15], Dynamic Practical Byzantine Fault Tolerance (Dynamic PBFT)[16], High Performance and Scalable Byzantine Fault Tolerance (HSBFT) [17] , Reputation Based Byzantine Fault Tolerance (RBFT) algorithm [18] .

And also instead of the methods and algorithms, there are some architectures as well in order to transform the BFT to fault avoidance.[21] Let's see some of the BFT solutions, their performance and how efficient those are.

C.3.1 OM Algorithm

In this solution, a commanding general must send an order to all the other generals. There are 02 conditions called interactive consistency conditions related to this algorithm ;

C1: All loyal lieutenants obey the same order
C2: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends

Since we require at least ⅔ loyal generals, if there is only one general or two generals, no consensus is required.
There is no solution for the scenario having only 03 generals including 1 traitor as well. Therefore we can decide there should be at least total 3m + 1 generals(including traitor) in order to cope with m traitors.[11]

Let's see why there is no solution for the 03 generals problem

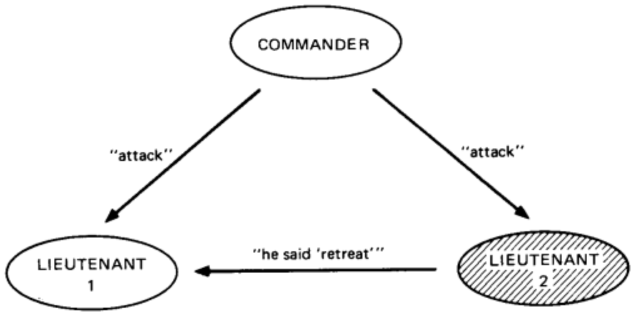

Figure 3: **03-general's problem (Lieutenant 2 a traitor)**

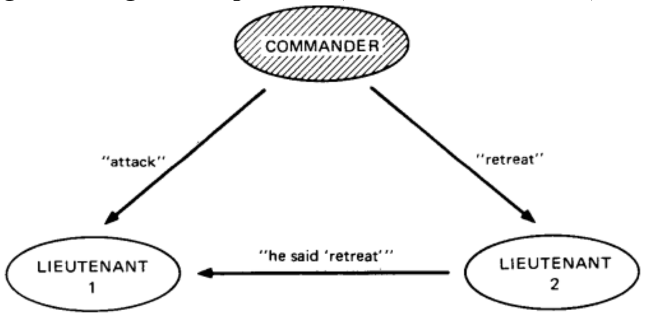

Figure 4: 03-general's problem (The commander a traitor)

In 'Figure 3', the commander is loyal, but Lieutenant 2(L2) is a traitor and sends a message to Lieutenant 1(L1) that he received as "retreat". In order to satisfy the C2, L1 must obey the commander.

And in the 'Figure 4', the commander is a traitor and sends two different messages to the Lieutenants. Therefore L1 doesn't know which message is correct. Hence, the above both scenarios are exactly the same to Lieutenant 1. Therefore there is no way for L1 to distinguish between these two situations, so he must obey the "attack" order in both of the scenarios.In the 'Figure 4', L2 must obey the "retreat" order while L1 obeys the "attack" order, which leads to violating the C1. Therefore, no solution exists for three generals' scenarios if one of them is a traitor.
Hence, let's see the explanation in OM(m) , when m=1 (m = traitors) where, there are 04 generals exists;

The scenario will be going as follows ;

- Commander will send the order to each lieutenant
- Each lieutenant record the message if received
- Then, each lieutenant will send the received message to others
- Finally, they will follow the majority order, comparing with the other's responses

m=1 and traitor is a Lieutenant:

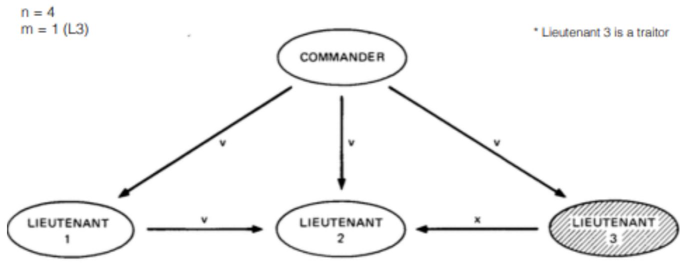

Figure 5: n = 4, m = 1

Commander will send 'v' to all, L1 sends the same value to L2 and L3(traitor) sends 'x' to L2.
L2 consists of 03 messages which are [v,v,x], therefore considering the majority L2 obtains the v which is the correct value.

m=1 and traitor is the commander:

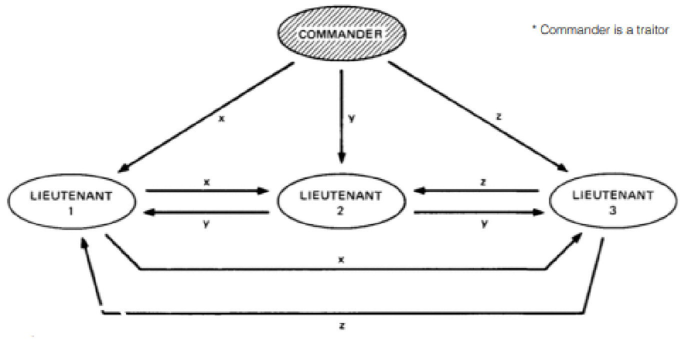

Figure 6: n = 4, m = 1, traitor = Commander [1]

Commander will send false messages to each lieutenant and each of them will send the message to others.
Therefore, all they obtain the same values, which are [x,y,z]. Hence, they will identify the commander as the traitor and retreat instead of attacking.

When the m=2, at least there should be 7 generals in order to cope with the 02 traitors. Therefore, when there are more nodes, there can be more malicious nodes and because of that, each node has to send more messages.

Problems with OM

Considering the above facts and as per the study, we can see this algorithm is performance wise very expensive when the nodes are increasing. And also the traitors can lie, and there is no solution with OM for less than 4 generals likewise there are further difficulties as well. Therefore this is not an efficient solution for the byzantine generals problem.

m	Messages sent
0	O(n)
1	O(n^2)
2	O(n^3)
3	O(n^4)

Table 1: m=number of traitors n=number of total generals

Because of the above mentioned problems, specialists have developed the Signed message algorithm (SM), a modified version of OM which works for any number of generals with 'm' number of traitors and the general's signature cannot be forged, and any modifications of the contents of his signed messages can be identified. [11]

However, the above-mentioned 02 algorithms (OM and SM) are the most earlier solutions and performance wise are very expensive, have many constraints in different scenarios, and there is no-point to use these OM and SM if all the nodes cannot communicate directly with each other, therefore it will lead us to a common question in the IT industry which is 'can we find a better solution?'

C.3.2 Practical Byzantine Fault Tolerance (PBFT)

In a distributed system, a primary node will be selected using PBFT and other nodes referred as secondary nodes.But the primary node can be changed time to time if there are any eligible secondary node exists in some scenarios such as primary node failures, change the primary node during pbft consensus round. The objective is to reach a consensus based on the majority rule.

There are three main aspects as follows: Normal-Case Operation, Garbage Collection, and View Change. [20]

- Normal-Case Operation - is to enable requests from clients to be executed in a certain order.
- Garbage Collection - periodically generates system log files for recovering in case of accident.
- View Change - will be executed when the primary node failures or becomes a byzantine replica and let those replicas elect a new primary

Here also, the number of malicious nodes must not be greater than or equal to ⅓ of all the nodes in the system. When the number of nodes increases, the system becomes more secure.

How the PBFT normal-case Operation is working:

Mainly, there are five phases, namely Request, Pre-Prepare, Prepare,Commit and Reply. If the master node fails or is attacked and does not work properly, the other backup nodes initiate a view
switch and select the new master node, waiting for the data synchronization to complete and then continue working [13]

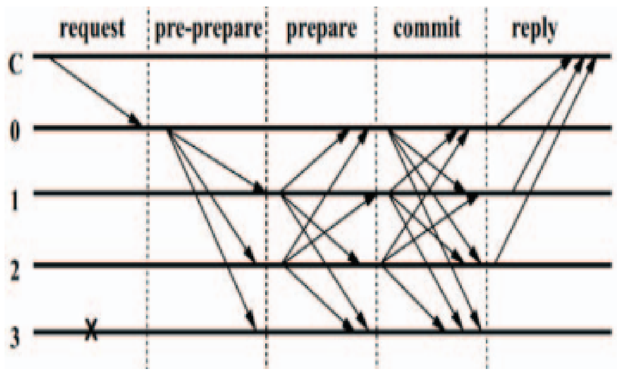

Figure 7: PBFT phases

The client sends a request to the primary node(0).Then, it will broadcast the request to all the secondary nodes(replicas), and all the replicas (1,2,3) carry out a process of pre-prepare and validation between them . Next, all the nodes perform the service request and return the result to the client. Since the maximum number of malicious nodes is m, the client must wait until received 'm+1' replies from different nodes in the system. To return the same result, we ensure the network reached a consensus. [19]

The primary node is changed during every PBFT consensus round and can be substituted by a view change protocol if a predefined amount of time has passed without the leading node broadcasting a request to the secondary nodes. And also, the majority of the honest nodes can replace the primary node as well by voting.

Problems with PBFT

In the research even it's mentioned that when the nodes are increasing, it is helpful for more security, there are performance,communication and network overheads with that. PBFT enabled distributed systems are vulnerable to sybil attacks, having response latency issues and cannot scale well due to its communication overhead. Further, slow down its use on a large scale[19] , PBFT's consensus protocol is complex, because replicas needs to communicate with each other in two phases, which leads to increase the message complexity [20]

Due to the various kinds of gaps in the OM,SM and PBFT as mentioned in the above, improved versions were developed based on different aspects which are more efficient solutions for the byzantine problems with less communication and network overheads and having better performances.

C.3.2 Improved versions of PBFT

- Threshold Signature Practical Byzantine Fault Tolerant(TS-PBFT) [13]
- Byzantine Fault Tolerant algorithm based on Vote algorithm (vBFT) [15]
- High Performance and Scalable Byzantine Fault Tolerance (HSBFT) [17]
- Reputation-based Byzantine Fault Tolerance (RBFT) algorithm [18]
- Scalable Efficient BFT protocol (SeBFT) [22]
- Byzantine consensus algorithm based on Gossip protocol (GBC), [13]
- Credit Practical Byzantine Fault Tolerance (CBPFT) algorithm, [13]

C.4 Critique on Byzantine Fault Tolerance

However, even though the PBFT solution was famous in the IT industry in the past, it has some issues such as poor scalability, high energy consumption and low efficiency. Therefore, many modified versions of PBFT algorithms were proposed. Some of these issues are still in the improved versions of the PBFT solutions as well. But in the vBFT, there is significantly improved performance when we consider the aspects of low-latency, energy consumption, dynamics and fault tolerance compared with consensus algorithms such as PBFT, CBPFT, and GBC algorithm. And also most of the current solutions need network connectivity highly in order to communicate with the nodes, therefore it will be difficult when the network expands further.

Therefore, cryptography based BFT methods were proposed which used the digital signatures. But, it needs a reliable infrastructure to pass their private and public keys, therefore if it fails, the total network will be failing. In order to overcome that issue, specialists in the industry developed non-cryptographic solutions.

In non-cryptography solutions, the main behavior is to communicate with each node or pair of nodes directly. Therefore, these solutions are hardly scalable. Hence, multi hop network solutions were introduced, where a node should depend on other nodes to broadcast a message.

D. Checkpointing

What is checkpointing

Checkpointing is one of the commonly used simplest techniques for fault tolerance. It represents the backward error recovery in fault tolerance. Mostly this was utilized for problems with the biggest and time-consuming application It is implemented in both hardware and software stack. Checkpoints are included in certain tasks and will save the intermediate state in secure storage. This state can be data, code or stack segment Whenever fault is detected at a specific checkpoint, it simply rolls back to the previous state without re-executing the same task from the beginning. This technique referred to the checkpoint and roll back recovery. Therefore, System should be restarted from the last checkpoint. Checkpoint schema can increase the execution time in faults. But it minimizes recovery time since there is no need to re-execute the whole task again. The cost of this mechanism is relatively higher than other methods as it utilizes a significant amount of CPU, I/O bandwidth. Other than that, it is rich in complexity. It is very important to recover the state as quickly as possible. By minimizing the size of a checkpoint it will speed up the recovery process as it may affect the overhead in the checkpoint. It can be solved by reducing the number of checkpoint occurrences in run time [23].When it comes to changing the state in most cases, it is better to minimize the checkpoint time. It is essential to minimize resource usage, restore time and power. Thus, the above characteristics must be exhibited in a good checkpointing mechanism.

Application-level vs System level check pointing

Checkpoint schema can be mainly classified into two variants. They are application level checkpointing and system level check pointing. According to the report provided by [24]. At the system level it is not required to change the code. State is saved completely and able to restart from the last checkpoint if it finds failure. However, on application-level data is checked hence it requires some programming effort. Therefore, the program might continue execution after fault because of less running time overhead. Developers of the application are responsible for the placement of the checkpoints. Users of parallel computing take the advantage of application level checkpointing. It allows users to decide when to make checkpoints without degrading the performance.

Coordinated Checkpointing

This is also known as synchronous checkpointing. Processes can take checkpoints in such a manner that always the resulting state is consistent. It comes across two phase structures. Through the first phase it processes uncertain checkpoints while in the second phase processes are permanent. So only one permanent checkpoint and at most one uncertain checkpoint needed to be stored. In case of any fault, a permanent checkpoint guarantees that state of the checkpoint would not be repeated. But an uncertain checkpoint is able to change to a permanent checkpoint. In ‘Figure 8’example (a) we can see process p0 sends message m to m1. If p1 receives a message before the checkpoint request which is sent by the checkpoint coordinator, there will be an unstable checkpoint. As depicted ‘Figure 8’ in example (b) we can solve this problem by avoiding every process to make a checkpoint prior to post checkpointing message. Coordinated checkpointing is domino free.

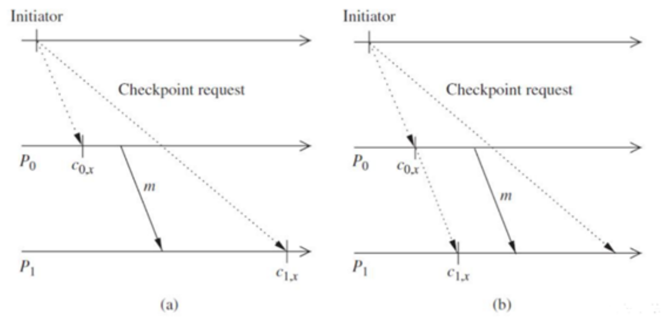

Figure 8: coordinated checkpointing

Uncoordinated Checkpointing

In the Uncoordinated checkpointing process they do not coordinate their activities. Each process stores its local checkpoint independently. System allows every process to decide the checkpoint. Therefore, the process will accept a checkpoint when it is well timed. It set up a consistent checkpoint by using dependencies. So, a stable global checkpoint is created by tracking relevant dependencies. As shown in “figure 9” , C12,C22,C32 checkpoints are not stable since stableness state consists of m4,m6 orphan messages. But states like C31, C21, C11 are consistent. The Domino effect is one of the major disadvantages of this approach.

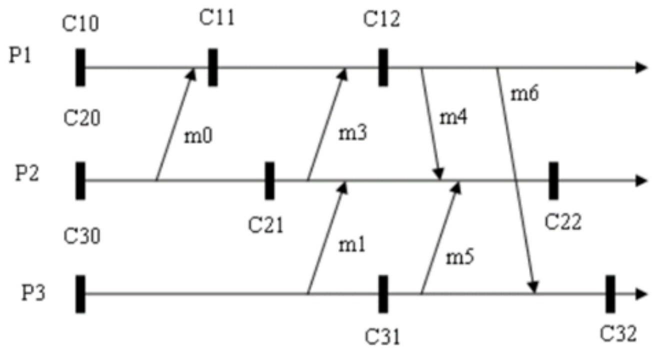

Figure 9: Un-coordinated checkpointing

Issues of Checkpointing

As we discussed earlier, the following are some common problems in checkpointing.

- Checkpointing overhead
- Orphan Message
- Domino effect
- Lost messages and live locks
- creating infinite number of rollbacks

How to overcome issues of checkpointing

1. Craft library

Checkpointing is still widely used in case of failures in software applications. [25] found CRAFT (Checkpoint-Restart and Automatic Fault Tolerance) which is implemented by C++. This approach is applied for application level checkpointing. CRAFT consists of primary and frequently used data types of checkpointing. The advantage of this library is it reduces the efficiency of overhead. However, lots of implementation effort is required.

2.Two states Checkpointing

Two state Checkpointing techniques are also used to minimize energy overhead and save a significant amount of time. This change in-between fault free and execution state. It is able to bear these two states. The first type is nonuniform intervals which are used when no fault has taken place. These intervals are committed to insert checkpoints in fault tree states. The other type is uniform intervals used from the time when the first fault happened. The abstract idea is to minimize the number of states even in the worst-case scenarios. Therefore, two states make sure to reduce execution time for faulty states and keep more time for energy management.

3. Synchronized Checkpointing and recovery

This approach avoids live lock by taking a consistent set of checkpoints. Live lock is a situation where processes block each other with the repeated state which results in not making any progress. In synchronized checkpointing checkpoints should be globally consistent and processes responsible to arrange their local checkpoints.

4.Cooperative checkpointing mechanism

Cooperative checkpointing basically decide when to avoid checkpoint requests made by system. It manipulates the global state in order to enhance performance and reliability. This approach makes use of the memory redundancy where multiple processes running parallel.

CRITIQUES ON CHECKPOINTING

The replication based fault tolerance technique is the most used method as per many journals. Undoubtedly it is one of the most efficient techniques , because it has been applied for many systems as discussed , even the critical ones , which shows how prominent this technique is. But in our point of view , we have seen that some drawbacks of this , such as high degree of replica, are not very much taken into attention in some applications , thus there is a doubt whether they are efficient as really mentioned. Comparing this to other techniques such as N- Version , in our point of view this is relatively simple with less algorithmic procedures , because algorithms can not be ensured to provide the correct result always.

In our point of view, N-version programming is not an ideal approach to tolerate faults in modern systems. It is time consuming and costly. And the biggest issue we see in N-version programming is assuming that individual groups will not make the same mistakes when developing individual versions. It actually does not matter what languages, tools, and environments we use to develop independent versions, it is always possible to have similar mistakes in the independent versions we generate.

In N-version programming, we use decision algorithms to select the version that gives the correct output. There we assume at least one independent version will give the correct output. But we can not guarantee that 100%. There can be scenarios where we will get wrong outputs from all the independent versions. The other issue we see in N-version programming is that it increases the complexity of the system. Earlier, when we discussed how we can apply N-version programming to Deep Neural Networks (DNN) [3], we saw how it increases the complexity of the system. All these issues make us question if it is worth applying N-version programming to our systems especially considering how time consuming and costly it is.

Most of the research papers evaluate that checkpointing technique is heavily used in mobile applications and cloud computing. This technique is also time consuming and much complex for developers. In our point of view checkpointing overhead is the main challenge to overcome. However, technicians intend to solve those problems by implementing new algorithms. They can manipulate the under-hood complexity in such a good manner. we suggest coordinated checkpointing rather than uncoordinated checkpointing as it has more advantages. Researchers still intend to make use of this technique by creating new libraries without considering the programming language. From the usage of a wide variety of developers we can recommend this technique for future works as well.

D. Conclusion

The applications of some of the fault tolerance techniques have been discussed throughout this journal. Replication , Checkpoint , N-Version , Byzantine are the techniques which have been discussed.Advantages , disadvantages and applications have been highlighted in each of them. Also the gaps of referred journals have been highlighted in the discussion itself. We have put forward our views regarding the topics in the “Critique” section. Fault tolerance being a characteristic that has become a must to systems , has got better and better with new discoveries. Also it will not stop evolving as there will be more complex systems in the future which will require more reliable and efficient fault tolerance techniques. Therefore we hope that from the ongoing researches , more and better fault tolerant techniques will be introduced for the betterment of technology.

References

[1] A. Ledmi, H. Bendjenna and S. M. Hemam, "Fault Tolerance in Distributed Systems: A Survey," 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), 2018, pp. 1-5, doi: 10.1109/PAIS.2018.8598484.

[2] Kumari, Priti, and Parmeet Kaur. "A survey of fault tolerance in cloud computing." Journal of King Saud University-Computer and Information Sciences 33.10 (2021): 1159-1176.

[3] Veerapandi, et al. “A Hybrid Fault Tolerance System for Distributed Environment using Check Point Mechanism and Replication.” International Journal of Computer Applications, vol. Volume 157, no. No 1, 2017, p. 48. Research Gate, https://www.researchgate.net/profile/Sumithra-Alagarsamy/publication/312508292_A_Hybrid_Fault_Tolerance_System_for_Distributed_Environment_using_Check_Point_Mechanism_and_Replication/links/5a4b60f90f7e9ba868b09e9d/A-Hybrid-Fault-Tolerance-System-for-Distri.

[4] C. Li, Y. Zhang, Y. Luo and G. Parr, "Adaptive Replica Creation and Selection Strategies for Latency-Aware Application in Collaborative Edge-Cloud System," in The Computer Journal, vol. 63, no. 1, pp. 1338-1354, Jan. 2020, doi: 10.1093/comjnl/bxz070.

[5] D. Zhou and Y. Tamir, "Fault-Tolerant Containers Using NiLiCon," 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 1082-1091, doi: 10.1109/IPDPS47924.2020.00114.

[6] C. Yuwei, "A Fault Tolerant Method for Network Distributed Flight Control System based on Task Replication," 2021 33rd Chinese Control and Decision Conference (CCDC), 2021, pp. 5074-5078, doi: 10.1109/CCDC52312.2021.9601528.

[7] R. Chen et al., "Replication-Based Fault-Tolerance for Large-Scale Graph Processing," in IEEE Transactions on Parallel and Distributed Systems, vol. 29, no. 7, pp. 1621-1635, 1 July 2018, doi: 10.1109/TPDS.2017.2703904.

[8] Randell, Brian. n.d. NATIONAL CONFERENCE ON NONLINEAR SYSTEMS & DYNAMICS, N-Version programming method of Software Fault Tolerance: A Critical Review

[9] H. Xu, Z. Chen, W. Wu, Z. Jin, S. -y. Kuo and M. Lyu, "NV-DNN: Towards Fault-Tolerant DNN Systems with N-Version Programming," 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2019, pp. 44-47, doi: 10.1109/DSN-W.2019.00016.

[10] V. Girdhar and E. Al-Masri, "N-Version Programming for Enhancing Fault Tolerance in Fog-based IoT Systems," 2020 6th International Conference on Science in Information Technology (ICSITech), 2020, pp. 109-114, doi: 10.1109/ICSITech49800.2020.9392033.

[11] Leslie Lamport, Robert Shostak and Marshall Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems

[12] Shubhani Aggarwal, Neeraj Kuma, The Blockchain Technology for Secure and Smart Applications across Industry Verticals, in Advances in Computers, 2021

[13] Wenxuan Jiang;Liquan Chen;Yu Wang;Sijie Qian, An efficient Byzantine fault-tolerant consensus mechanism based on threshold signature, 2020 International Conference on Internet of Things and Intelligent Applications (ITIA) - 2020

[14] Soohyeong Kim;Sejong Lee;Chiyoung Jeong;Sunghyun Cho, Byzantine Fault Tolerance Based Multi-Block Consensus Algorithm for Throughput Scalability, 2020 International Conference on Electronics, Information, and Communication (ICEIC), Year: 2020

[15] Haiyong Wang;Kaixuan Guo, Byzantine Fault Tolerant Algorithm Based on Vote, 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Year: 2019

[16] Xu Hao;Long Yu;Liu Zhiqiang;Liu Zhen;Gu Dawu, Dynamic Practical Byzantine Fault Tolerance, 2018 IEEE Conference on Communications and Network Security (CNS), Year: 2018

[17] Yanjun Jiang;Zhuang Lian, High Performance and Scalable Byzantine Fault Tolerance, 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), year 2019

[18] Kai Lei;Qichao Zhang;Limei Xu;Zhuyun Qi, Reputation-Based Byzantine Fault-Tolerance for Consortium Blockchain, 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS) , 2018

[19] Seybou Sakho; Jianbiao ZHANG; Firdaous Essaf; Khalid Badiss; Tchewafei Abide; Julius Kibet Kiprop, Research on an improved practical byzantine fault tolerance algorithm, 2020 2nd International Conference on Advances in Computer Technology, Information Science and Communications (CTISC), Year: 2020

[20] Li Zhang ;Qinwei Li, Research on Consensus Efficiency Based on Practical Byzantine Fault Tolerance, 2018 10th International Conference on Modelling, Identification and Control (ICMIC), 2018

[21] Noor O. Ahmed; Bharat Bhargava, From Byzantine Fault-Tolerance to Fault-Avoidance: An Architectural Transformation to Attack and Failure Resiliency, IEEE Transactions on Cloud Computing, 2020

[22] Yanjun Jiang; Zhuang Lian, Scalable Efficient Byzantine Fault Tolerance, 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Year: 2019

[23] S. Muhammad Abrar Akber, H. Chen, Y. Wang and H. Jin, "Minimizing Overheads of Checkpoints in Distributed Stream Processing Systems," 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), 2018, pp. 1-4, doi: 10.1109/CloudNet.2018.8549548.

[24] J. Posner, "System-Level vs. Application-Level Checkpointing," 2020 IEEE International Conference on Cluster Computing (CLUSTER), 2020, pp. 404-405, doi: 10.1109/CLUSTER49012.2020.00051.

[25] F. Shahzad, J. Thies, M. Kreutzer, T. Zeiser, G. Hager and G. Wellein, "CRAFT: A Library for Easier Application-Level Checkpoint/Restart and Automatic Fault Tolerance," in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 3, pp. 501-514, 1 March 2019, doi: 10.1109/TPDS.2018.2866794.