IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

Fault Tolerant Mechanism in Blockchain-Survey

Dr. S. Danraj[1], Rohit N[2], Hiban Parol[3], Sreeshnav K[4], Ranjana C[5]
Director, EPA, Coimbatore[1]
Students, Department of Computer Science & Engineering[2,3,4]
Assistant Professor, Department of Computer Science & Engineering[5]
Vedavyasa Institute of Technology, Malappuram, Kerala, India

Abstract: *The purpose of fault tolerance system is to avoid a failure of the overall system in spite of existing faults in the different components. Redundancy has been a significant technique to assure fault tolerance design in the digital system. Among several redundancy techniques, hardware redundancy is constantly used to improve the reliability of the digital systems. While active hardware redundancy is useful to detect faults and recovery, passive hardware redundancy is useful to hide faults in hardware components. Triple modular redundancy is a passive hardware redundancy technique where faults are hidden, and only correct data are passed as output from the system.*

Keywords: Block chain

I. INTRODUCTION

Fault tolerance is one of the critical issues in Wireless Sensor Network (WSN) applications. The problem of Missing sensor node, communication link and data are Inevitable in wireless sensor networks. WSNs experience failure Problems due to various factors such as power depletion, environmental impact, radio interference, asymmetric Communication links, dislocation of sensor node and collision Blockchain fundamentals are based on a distributed Peer-to-peer network, which has to deal with fault tolerances, Like all other similar networks. This is specifically important for blockchain technology, due to Its promised data integrity Features like immutability and traceability Application partitioning and code offloading are being researched Extensively during the past few years. Several frameworks for Code offloading have been proposed. However, fewer works Attempted to address issues occurred with its implementation in Pervasive environments such as frequent network disconnection Due to high mobility of users. The fault tolerant algorithm can easily adapt to different execution Conditions while incurring minimum overhead.

II. FAULT TOLERANT

It refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail. The objective of creating a fault tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems. Fault-tolerant systems use backup components that automatically take the place of failed components, ensuring no loss of service. These include:

1. Hardware system
2. Software system
3. Power source

2.1 Hardware System

Hardware systems that are backed up by identical or equivalent systems. For example, a server can be made fault tolerant by using an identical server running in parallel, with all operations mirrored to the backup server.

2.2 Software System

Software systems that are backed up by other software instances. For example, a database with customer information can be continuously replicated to another machine. If the primary database goes down, operations can be automatically redirected to the second database.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

2.3 Power Sources

Power sources that are made fault tolerant using alternative sources. For redundancy based Redundancy is one of the efficient techniques to achieve fault tolerance mechanism in the digital system. As its name suggest, redundancy is basically an addition of hardware and software resources, information or time in a system more than it required to perform their normal operation. Many organizations have power generators that can take over in case main line electricity fails.

2.4 Advantages of Fault Tolerant

1. A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.
2. The more complex the system, the more carefully all possible interactions have to be considered and prepared for.
3. A fault-tolerant design may allow for the use of inferior components

2.5 Redundancy Based

It is one of the capable strategies to achieve variation to non-basic disappointment instrument in the modernized system. As its name software resources, information or time in a structure an overabundance to play out their customary movement. There are four overabundance strategies and they are:

1. Programming Redundancy
2. Time Redundancy
3. Information Redundancy
4. Hardware Redundancy

A. Software Redundancy

The physical defects in the hardware components can barely re-occurred once it is discovered and repair, fixing bugs in the software programs will create greater chance to create other errors in the coding. Despite the presence of different software development process, there are chances to develop error prone software due to novice developers, lack of testing or inadequate time and money for the full phase development of the software. N-version programming is one of the software fault-tolerance technique where a program writes for N times and execute in parallel to take majority output as a final output of the program.

B. Time Redundancy

The concept of time redundancy is to run the program multiples times in the presence of same hardware configuration and compare the produce results. It reduces the expense on expensive hardware addition and also avoid parallel execution of the programs. Since a program can be run multiple times in the same modules and compare the results to identify errors or faults, it will efficient compared hardware and software redundancy technique. Besides that, it is suitable for transient or intermittent faults as a frequent run of programs in same module will help to get majority output as the actual output of the system. However, one of the considerable disadvantages of time redundancy is the requirement of large amount of time to identify the faults and defects in the system compared to hardware and software system

C. Information Redundancy

In Here, there will be an addition of extra information along with data to ensure to integrity of the information changes during a storage or transmission. An error detecting codes and correcting codes, and self-checking circuits are popular mechanism for information redundancy. Parity code is widely used for error detection in the memory of the computer system. A parity bit is generated by a parity generator and data is encoding through computation of its parity. When there Is changes in the computed parity bit with stored parity bit, there is an indication of data changes and error signal is sent to the processor of invalid memory data. On the other hand, data which are encoded with error-correcting

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

codes contains both errors and adequate redundancy to recover the desired data. Meanwhile, self checking circuit will produce valid output word when there is valid input and when there is existence of fault, it will produce invalid output code to detect the fault in the system

D. Hardware Redundancy

It can be achieved through the addition of extra hardware to the system. As the hardware components are getting cheaper with advancement of technology, it can be considered as suitable mechanism to achieve reliability in the system. Apart from that, it also does not require continue observation and will not take more time to identify and mask the error compared to other redundancy techniques For example, an addition of processors, data or memory buses, power or even memories can easily achieve hardware redundancy. There are commonly three approaches to obtain hardware redundancy techniques and they are as follow

1. Passive Redundancy
2. Active Redundancy
3. Hybrid Redundancy

1. Passive Redundancy

It is responsible for hiding and covering the faults in the hardware components rather than detecting those faults. It will produce the result based on polling mechanism and provide the correct output from the system despite the existing of faults in the components. When there are multiples faults than polling circuit can cover, then it cannot hide the faults and failure of the system is imminent. Triple modular redundancy and N modular redundancy are the suitable example of fault tasking technique through use of redundant hardware in the system

2. Active Redundancy

It is used to detect the faults in the components and recover from those faulty components. There are use of different techniques for fault detections and computation with duplication is one of the techniques where two duplicate modules execute identical computation in parallel and with use of comparator results are compared. If the results are not equal between two modules, an error signal will be produced

3. Hybrid Redundancy

It Includes features from both earlier techniques, is another hardware redundancy technique to identify faults in the components and recover from those faults. One of the approaches of hybrid redundancy technique is duplex-triplex architecture where two duplication with comparison technique along with TMR is used to mask the errors, detect the faults and remove those faults from the system to produce desirable output. One of the major disadvantages of hybrid redundancy is that the methods which are used to implement it are costly

III. Triple Modular Redundancy

Triple Modular Redundancy is mainly based on redundancy of hardware components of the system. It is highly used to make system more reliable and continue to perform their operations against the soft errors. When an error occurs in sequential circuits, which indicates to the different storage in the system such as registers, memories, flip-flips and counters, there will be a change in the saved state in different storage and lead to the execution of the program different from the expected one. To minimize effects of soft errors, TMR is designed in microprocessors so that errors will not halt the flow of the program. The generalization of TMR technique in N modular redundancy technique. While there is presence of n modules in the N modular redundancy technique, TMR will have 3 identical functional hardware units to perform the operation. The concept of the TMR design is to use three duplicate modules which will take same input. All the operation within the module is same for all three of those modules, so, the use of input data is same of all three cases. On the top of the identical modules, there is apply of voting mechanism to get majority output as the actual output of the system. So, voter circuit will take all three outputs of the hardware unit as the input to the unit and majority of input will consider as the actual output of the system. The basic purpose of Triple Modular Redundancy is

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

to mask the errors exist in the functional unit of the system. So, it can be considered as the passive hardware redundancy technique to achieve the reliability of the system

We are concluding that Hardware redundancy is useful to improve the reliability of the system. Although the structure of triple modular redundancy is simple and cost effective, a single point failure i.e. voter circuit could cause the failure of the overall system. Meanwhile, different techniques were proposed to improve the reliability of the triple modular redundancy, however, many of these techniques were based on assumptions that no two modules will fail at the same time and voter circuit will never fails. The report has performed detail analysis on some techniques which will be useful to improve the voter circuit in the system.

IV. Analysis of Fault Tolerance in Permissioned Blockchain Networks

Blockchain fundamentals are based on a distributed peer-to-peer network, which has to deal with fault tolerances, like all other similar networks. This is specifically important for blockchain technology, due to its promised data integrity features like immutability and traceability. In this paper, we analyze the basic principles of distributed consensus algorithms while focusing on permissioned blockchain networks. We analyze how distributed consensus mechanisms in two permissioned blockchain networks cope with a crash fault-prone environment. The purpose and usage of blockchain technology differ from case to case. For this reason, blockchain networks, as the environment which enables the features of the related technology, are generally classified into two main categories, depending on the desired network type, i.e. permissioned and permission less blockchain.

In permissionless blockchain networks (e.g., cryptocurrency networks like Bitcoin, Litecoin, Ethereum, etc.), everyone can join as an equal participant (referred as a node) of the network. The nodes are unknown to each other due to the public and permissionless nature; consequently, a trustless environment is set up. However, nodes in the permissioned blockchain networks (e.g., Ripple, Hyperledger Fabric) are identified to each other, since these are made up of a consortium or a group of somehow related entities. Nevertheless, nodes in the permissioned network have potentially various roles and permissions assigned, which are not necessarily equal. Mutual trust among part of nodes may already be built, but this is not necessary for all nodes of the network.

V. Related Works

Minjeong et al. analyzed the security of Stellar consensus protocol, which implements the federated Byzantine agreement (FBA). The authors evaluated how the impact of the cascading node failure impacts the overall fault tolerance of the Stellar blockchain network.

Ji et al. proposed permissioned blockchain-based personal information management system, which, as distributed consensus protocol uses the CFT-based Raft algorithm. They performed an evaluation of the network segmentation stability of blockchain network based on the proposed system.

Hao et al. described a novel Dynamic PBFT protocol, which allows that participants of consensus to dynamically join and exit the network. In the paper, a CFT-based fault tolerance analysis of proposed consensus protocol was performed. Most of the published papers like provide a scalability and performance analysis of the distributed consensus algorithms used in different types of blockchain networks. For the best of our knowledge, no other contribution provides a CFT analysis, specifically focused on the most popular permissioned blockchain projects.

VI. Permissioned Blockchain Project

According to Forbes, among the fifteen big companies which are actively exploring blockchain technology, Hyperledger is the most popular permissioned blockchain project family. Hyperledger is an open-source project hosted by The Linux Foundation, founded to advance cross-industry blockchain technologies. At the moment, Hyperledger hosts five blockchain-based frameworks, i.e. Burrow, Fabric, Indy, Iroha, and Sawtooth. Hyperledger Burrow is listed as the project in the incubation stage, while all other frameworks are listed as active (production-ready). Because of reasons above, we have decided to analyze the crash fault tolerance of two production-ready permissioned blockchain platforms under the Hyperledger blockchain frameworks umbrella. Fabric was chosen, for being the first production-ready project, while Hyperledger Iroha for being the newest and latest announced production-ready Hyperledger

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

framework. Hyperledger Fabric provides a modular and extensible system for establishing a permissioned blockchain network, initially contributed by Digital Asset and IBM. It is the first blockchain framework which enabled the usage of general purpose language (e.g., Go, Node.js, Java) for implementing blockchain-based smart contracts, called Chain codes. Fabric also supports pluggable consensus protocols. At the moment of writing, Fabric supports three different distributed consensus mechanism implementations, i.e. Solo, Kafka, and Raft. None of the supported distributed consensus mechanism implementations support the BFT property. Kafka and Raft support the CFT property, while Solo does not offer any fault tolerance. Distributed consensus algorithms are the core of Fabric ordering service, the goal of which is to provide the final and correct order of transactions in a deterministic fashion. Fabric users can establish specific channels, which purpose is to provide visibility masking of chosen transactions only to a subset of participants.

VII. Distributed Consensus

There are many CFT-based distributed consensus mechanism implementations, e.g. Paxos, Kafka, Zookeeper and Raft, but only the latter is presented in this paper, due to being the first step towards a full BFT-based consensus algorithm in Hyperledger Fabric. Similarly, YAC was selected among many BFT-based consensus algorithms

7.1 Raft

Raft is a CFT-based distributed consensus mechanism implementation, making it immune only to crashed nodes and not to the Byzantine ones. Managing consistency of a replicated log among the nodes, even some of them being offline, is its primary goal. Raft was heavily inspired by one of the first distributed consensus mechanism implementations, i.e. Paxos, however its design is simplified and more efficient.

Raft is categorized as a voting-based distributed consensus. Each node, in any given time, is assigned with one of the following three roles: leader, candidate or follower. The leader is responsible for the communication with the clients and is the only node, which has the ability to make changes to the log. There can be only one leader at any given time. The candidate is the node, which has the opportunity to become the next leader. The follower is every other node, which has not been selected as a leader, and its only job is to follow the leader.

7.2 YAC

YAC is a BFT-based distributed consensus algorithm, making it immune to the Byzantine or malicious nodes. According to a distributed computer system can never handle more than f Byzantine nodes in a system of $3f + 1$ nodes. YAC was released alongside Iroha, and its structure is heavily adjusted for the architecture of Iroha.

The consensus flow is divided into rounds. The pipeline of a round consists of the three parts: (1) client sending commands, (2) an ordering phase and (3) a collaboration phase. In the first phase, a client constructs the transaction out of commands and sends it to a node, where it is handled by its ordering service. In the ordering phase, the received transactions are ordered and combined into block proposals. Block proposals are forwarded to other nodes, initiating the collaboration phase. Nodes check the validity of the transactions in the block proposal and create the new block out of valid transactions. After that, the new block is sent to the other nodes with the block proposal's hash and the block's hash. Nodes then vote, whether they accept/commit or reject the block. More than two-thirds of all nodes must commit the block that it is accepted and added to the blockchain.

We are concluding that In this paper, we set up to identify, which distributed consensus mechanisms are used in permissioned blockchain networks. We came to the conclusion that by design, the majority of such networks support voting-based distributed consensus algorithms, which however provide mainly the CFT property, while not completely avoiding BFT. However, due to the trustful nature of permissioned blockchain networks, some authors claim that it is more important to solve the crash-based faults than Byzantine ones. We furthermore analyzed how the distributed consensus mechanisms, in the aforementioned networks, cope with a crash fault-prone environment.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

VIII. Fault Tolerant Mechanism for Partitioning and Offloading Framework in Pervasive Environments

Application partitioning and code offloading are being researched extensively during the past few years. Several frameworks for code offloading have been proposed. However, fewer works attempted to address issues occurred with its implementation in pervasive environments such as frequent network disconnection due to high mobility of users. Thus, in this paper, we proposed a fault tolerant algorithm that helps in consolidating the efficiency and robustness of application partitioning and offloading frameworks. To permit the usage of different fault tolerant policies such as replication and checkpointing, the devices are grouped into high and low reliability clusters. Experimental results shown that the fault tolerant algorithm can easily adapt to different execution conditions while incurring minimum overhead.

A mobile pervasive environment consists of users interacting with mobile devices connected to stationary devices, desktops, servers or other mobile devices wirelessly. Due to mobility of users, frequent network disconnections have become a normal characteristic, and as a consequence, this results in failure of any mobile distributed system running in such environment affecting negatively the reliability of the latter. Several fault tolerance mechanisms have been proposed to solve the reliability problem in distributed computing systems. Almost all proposed techniques consider environments with wired homogeneous computational devices. Thus, they are difficult to adapt in a wireless heterogenous mobile computing environment as opposed to grid computing systems. Fault tolerance is a process of reinstating the normal or an acceptable behavior of a system. In pervasive computing environments, network disconnection in the middle of the execution of a task is frequent. It is due to, mainly, mobility of users. For example, if user A's device act as a participating device in a cluster of devices collaborating to execute an application partitioned and offloaded from user B's device and the former moves away from the cluster, then a failure is generated. Hence, this paper proposes a fault tolerant algorithm, using reactive fault tolerant methods, which is an independent component that can be added to any offloading framework. The fault tolerant component takes as input the different tasks offloading schedules from the existing offloading systems and ensure the complete application execution by the application of different fault tolerant policies.

IX. Related Works

Two reactive fault tolerance mechanisms often used are checkpointing and replication. Using checkpointing, snapshot of an application state is taken at a pre-define time interval and the latter is saved on disk. The system reliability is determined by the time elapsed between two checkpoints. However, in the case of replication, no snapshot of application state is saved. Actually, a replication of the application is executed in parallel on other computational devices to ensure complete processing of task.

The paradigm of distributed computing encompasses grid computing, mobile grids, cluster computing among many others. In such systems, the computational resources loosely coupled are connected by means of a network that requires the management of fault tolerance to ensure the system stability, robustness and reliability. The authors in [6] made a performance comparison between dynamic load balancing (DLB) and job replication (JR) on distributed systems robust level. A measure statistic Y and a corresponding threshold value Y* were provided, such that DLB consistently outperformed JR, and the reverse is true while Y<Y*. In the authors proposed an incremental checkpoint and restart model for high performance computing (HPC). So as to minimize the overhead of checkpointing, the method performs a set of incremental checkpoints between two full checkpoints by only saving the address space that has changed since the last checkpoint. However, the fault tolerance mechanism in distributed computing systems only take into consideration failures of devices as the wired networks are stable. But when considering mobile pervasive environments whereby computational devices are mostly mobile and networks are mostly wireless, then existing mechanism fall short. A replication-based algorithm that uses the Weibull distribution for mobility analysis was proposed to approximate the number of replicas so as to maintain a level of fault tolerance for mobile grid systems. A onefold fault tolerance policy is applied by most existing algorithms which is not reliable for resource-limited mobile devices collaborating in smart mobile spaces as there is no assurance for the availability of computational devices.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

X. Proposed Fault Tolerant Algorithm

The fault tolerant mechanism along with other components to enable the offloading process is depicted in Figure 2. The interaction diagram of the fault tolerant component along with the code offloading engine, source device and participating devices is shown in Figure 3. The offloading decision making component output a list of tasks to be offloaded and their corresponding host devices. We refer to this output as the task offloading scheduling plans (TOSP). The device clustering module clusters the participating devices and the policy generator binds the relevant fault tolerance policy to each task after decoding the TOSP. Three criteria are considered. They are the performance of the device, its availability and its data transfer speed within the network

XI. Evaluation

11.1 The Setup

A number of simulations are performed so as to evaluate the performance of the proposed fault tolerant algorithm. Three metrics are considered for this evaluation and they are the application completion time, the fault tolerant algorithm overhead cost and the number of control messages for fault tolerance. Conventional fault tolerant algorithms are compared with the results obtained to better situate the performance of the proposed algorithm. A simulator is implemented based on the INET Framework to simulate the devices in the mobile network. INET is an open-source model suite for wired, wireless and mobile networks running on top of OMNeT++ which is an event discrete simulator. Different topologies with varying bandwidth and delay values for each communication link are generated. Table 2 and 3 describes the task parameters list and device parameters list for the simulation respectively.

Parameters	Value
Number of applications	50
Length of computation (no of instructions)	20000 – 100000
Size of data (MB)	0.5 – 10

Table 2: Task parameter values for the simulation

Parameters	Value
CPU Speed (MIPS)	1000 – 100000
Total time available (s)	1000 – 30000
Weibull (shape & scale) (failure point)	α 1.21, β 94.08
Bandwidth WIFI (MBps)	0.9 – 1.2
Number of devices	20 – 50

Table 3: Device parameter values for the simulation

We designed and implemented a π-calculator to estimate to a certain extent the value of π. The workloads used in this simulation consists of a set of applications represented by different randomly generated DAGs. A task of an application is represented by a vertex in a DAG. Each task has to calculate the value of π and takes as input the number of times to run the approximations and the number of decimal places desired.

Each vertex has also two values associated with it, that is, the amount of computation and data size. The two values are generated from within a specified range. The device parameters listed in Table 3 shows that the latter's processing speed is measured in Million Instructions Per Second (MIPS). With every tick of the clock, the CPU fetches and executes one instruction. The clock speed is measured in cycles per second, and one cycle per second is known as 1 hertz. This means that a CPU with a clock speed of 2 gigahertz (GHz) can perform two billion cycles per second.

To simulate failures of devices, a 2-parameter Weibull distribution is considered to generate random time between failures and the failure time points are computed by the device start time to the time generated from the distribution. And for realistic mobility patterns, we used the CRAWDAD trace dataset [17] to get the shape and scale parameters of the Weibull distribution.

The simulator, see Figure 4, consists of several modules. Simulation configurations are stored in the simulation settings database or file. And the latter is accessed by the application generator and device generator to obtain simulation data. Random application directed acyclic graphs (DAGs) are generated by the application generator that

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

simulates the mobile tasks and their dependencies. Using the device information from the simulation settings module, the device generator output data for mobile devices. And the device cluster generator uses the former along with the network generator to generated the high and low reliability clusters.

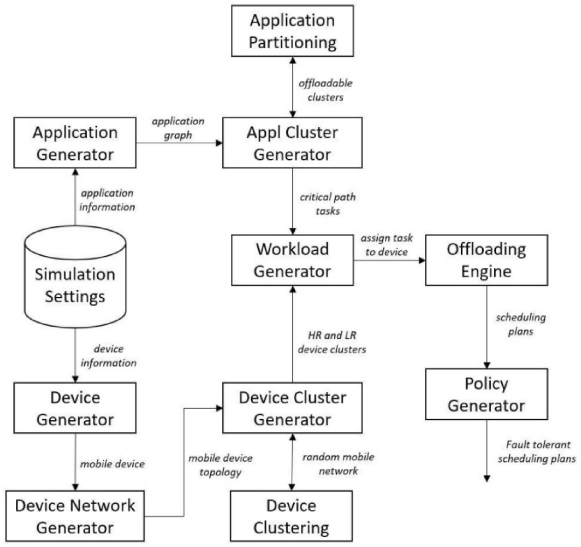

Figure 4: The structural design of the simulator

11.2 Results and Analysis

The weight factors Y 0.2, Z 0.6 and λ 0.2 are used for the experiments. We evaluated the performance of the proposed fault tolerant algorithm with 500 randomly generated applications DAGs. Each application graph is generated with random number of edges and vertices. Each task is bind with a data size and the amount of computation. The output is then compared to other basic fault tolerant strategies such as checkpointing only policy C^*, replication only policy R^* and lastly with a no-fault tolerant policy $NoFT^*$, that is, no fault tolerance policy is applied to the scheduling plans. Two performance experiments are considered. Experiment 1 analyze the effect of device availability reflecting on its median time between failures (MTBF). Experiment 2 assesses the consequence of task computation on the fault tolerant algorithm's performance.

XII. EXPERIMENT

In this experiment, the fault tolerant algorithms performance is evaluated. The MTBF is used to denote the availability of the device. The Weibull distribution is used to generate the failure time for each device. Figure 5 shows the average application completion time. We can see that the proposed fault tolerant algorithm outperformed the other three strategies.

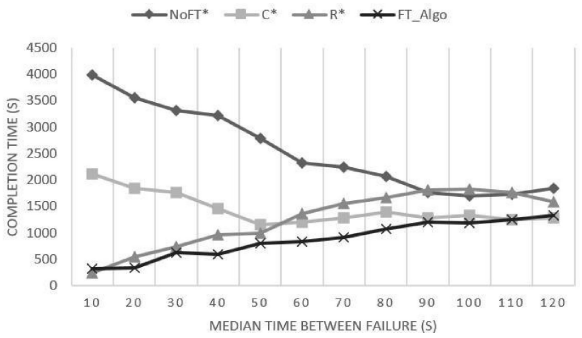

Figure 5: Application completion time based on different device availability

Notice when the MTBF is small (for example, 10 and 20), it implies that the availability of devices is very high and the $NoFT^*$ strategy results in the worst completion time of 4000 seconds compared to other. Whereas R^* and the proposed

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

fault tolerant algorithm (FT_Algo) strategies have amongst the lowest completion time. This is because when the failure occurs more often, the other two strategies, that is, C^* and $NoFT^*$ keeps restarting the task execution and that results in a higher overall application completion time than R^* and FT_Algo. Also, when the MTBF is low, the FT_Algo applies the replication policy thus the result is similar to the R^* strategy. When the MTBF is above 90, all strategies seem to generate stable results. It is because the devices are more and more reliable, the R^* strategy generates more redundancy overhead for transmitting the replica than C^* as depicted in Figure 6.

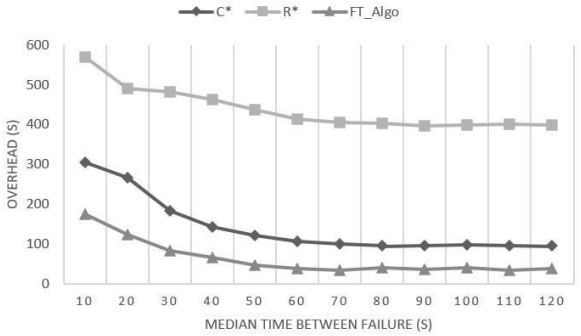

Figure 6: Overhead cost based on different device availability

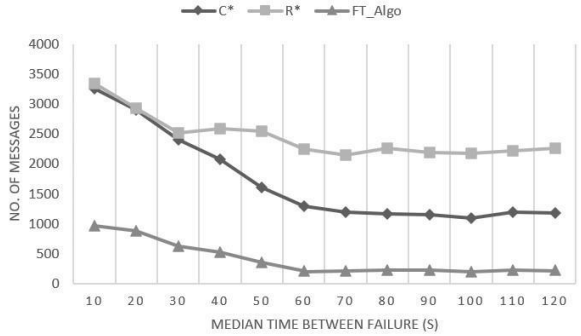

Figure 7: Number of messages based on different device availability

As the MTBF increases, this implies, more time is available for the tasks to be completed before failures occur. Hence, a decrease in the overhead for all strategies. As illustrated in Figures 6 and 7, the overhead and number of messages generated by the overhead tends to decrease and stabilize, which is in line with the overall application completion time.

XIII. Conclusion

The mobility of users makes the mobile partitioning and offloading system susceptible to failures. Since most existing fault tolerant algorithms for distributed systems concentrates mainly on device crash failures, their adaptability to pervasive environment that consists of frequent wireless network failures seem difficult and inadequate. Thus, in this paper, we presented a fault tolerant algorithm that helps in consolidating the efficiency and robustness of the partitioning and offloading frameworks. To permit the usage of different fault tolerant policies such as replication and checkpointing, the devices are grouped into high and low reliability clusters. Experiments result shown that the fault tolerant algorithm can easily adapt to different execution conditions while incurring minimum overhead.

References

[1]. F. J. Atero, J. J. Vinagre, J. Ramiro, and M. Wilby, "A low energy and adaptive routing architecture for efficient field monitoring heterogeneous wireless in sensor networks," In: 31st Int. Conf. on Distributed Computing Systems Workshops, IEEE, pp.172 – 181, 2011.

[2]. C. Wu and Y. Chung, "Heterogeneous Wireless Sensor Network Deployment and Topology Control Based on Irregular Sensor Model," Adv. Grid Pervasive Comput. Springer Berlin Heidelberg, pp. 78–88, 2007.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

[3]. G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," Ad Hoc Networks, vol. 7, no. 3, pp. 537–568, 2009.

[4]. S. Climent, J. V. Capella, N. Meratnia, and J. J. Serrano, "Underwater sensor networks: A new energy efficient and robust architecture," Sensors, vol. 12, no. 1, pp. 704–731, 2012.

[5]. M. Sudip, S. Chandra Misra, I. Woungang, Guide to Wireless Sensor Networks. London: Springer, 2009.

[6]. R. Ma, L. Xing, and H. Michel, "Fault-Intrusion Tolerant Techniques in Wireless Sensor Networks," In: 2nd Int. Symposium on Dependable, Autonomic and Secure Computing, IEEE. pp. 85-94, 2006.

[7]. C. Zhang, J. Ren, C. Gao, Z. Yan, and L. Li, "Sensor fault detection in wireless sensor networks," IET Int. Commun. Conf. Wirel. Mob. Comput., pp. 66–69, 2009.

[8]. B. Khelifa, H. Haffaf, M. Madjid, D. Llewellyn-Jones, "Monitoring Connectivity in Wireless Sensor Networks," J. of Future Generation Communication and Networking, vol. 2, no. 2, pp. 507 – 512, 2009.

[9]. E.Dubrova, Fault Tolerant Design: An Introduction. New York: Springer, 2013.

[10]. M. Younis, I. F. Senturk, K. Akkaya, S. Lee, and F. Senel, "Topology management techniques for tolerating node failures in wireless sensor networks: A survey," Comput. Networks, vol.58, no.1, pp.254-283, 2014.

[11]. L. Moreira, H. Vogt, M. Beigl, A survey on fault tolerance in wireless sensor networks. braunschweig, Germany: Sap research, 2007.

[12]. S. Kim, J. Ko, J. Yoon, and H. Lee, "Multiple-Objective Metric for Placing Multiple Base Stations in Wireless Sensor Networks," In: 2nd Int. Symposium on Wireless Pervasive Computing, IEEE. pp. 627-631, 2007.

[13]. R. Raj, M. Ramesh, S. Kumar, "Fault Tolerant Clustering Approaches in Wireless Sensor Network for Landslide Area Monitoring." In: Int. Conf. on Wireless Networks, pp.107-113, 2008.

[14]. H. Ammari, S. Das, "Fault tolerance measures for large-scale wireless sensor networks," ACM Trans. Auton. Adapt. Syst., vol. 4, no. 1, pp. 1– 28, 2009.

[15]. H. Alwan and A. Agarwal, "A Survey on Fault Tolerant Routing Techniques in Wireless Sensor Networks," 2009 Third Int. Conf. Sens. Technol. Appl., IEEE, p. 366-371, 2009.

[16]. D. Ioan Curiac, C. Volosencu, D. Pescaru, L. Jurca, A. Doboli, "Redundancy and its applications in wireless sensor networks: a survey," WSEAS Trans. on Computers, vol. 8, no.4, pp.705-714, 2009.

[17]. Q. Liang, "Fault-Tolerant and Energy Efficient Wireless Sensor Networks: A Cross-Layer Approach," In: IEEE Military Communications Conference, IEEE. pp.1862-1868, 2005.

[18]. M. Lee, Y, Choi, "Fault detection of wireless sensor networks," J. of Computer Communications, vol.31, no.14, pp.3469–3475, 2008.

[19]. M. Qiu, J. Liu, J. Li, Z. Fei, Z. Ming and E. Sha, "Informer homed routing fault tolerance mechanism for wireless sensor networks," J. of Systems Architecture, vol.59, pp.260-270, 2013.

[20]. S. Halder, M. Mazumdar, P. Chanak, I. Banerjee, "FTLBS: Fault Tolerant Load Balancing Scheme in Wireless Sensor Network," J. of Advances in Computing and Inform. Technology, pp.621-631, 2012.

[21]. I. Korbi, Y. Ghamri Doudane, R. Jazi, L. Azouz Saidane, "CoverageConnectivity based Fault Tolerance Procedure in Wireless Sensor Networks," In: 9th Int. Conf. on Wireless Communication and Mobile Computing, IEEE. pp. 1540-1545, 2013.

[22]. Y. Lai and H. Chen, "Energy-efficient fault-tolerant mechanism for clustered wireless sensor networks," In: Int. Conf. Comput. Commun. Networks, IEEE, pp. 272–277, 2007.

[23]. N. Bansal, T. Sharma, M. Misra, R. Joshi, "FTEP: A Fault Tolerant Election Protocol for Multi-level Clustering in Homogeneous Wireless Sensor Networks," In: 16th Int. Conf. on Networks, IEEE. pp.1-6, 2008.

[24]. A. Kaur and T. Sharma, "FTTCP: Fault Tolerant Two-level Clustering protocol for WSN," Int. J. on Networking Security, vol.1, no.3: pp.2833, 2010.

[25]. L. Karim, N. Nasser, and T. Sheltami, "A Fault Tolerant Dynamic Clustering Protocol of Wireless Sensor Networks," Glob. Telecommun. Conf., IEEE, pp. 1–6, 2009.

[26]. A. Bari, A. Jaekel, J. Jiang, and Y. Xu, "Design of fault tolerant wireless sensor networks satisfying survivability and lifetime requirements," Comput. Commun., vol.35, no.3, pp. 320-333, 2012.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 12, Issue 3, December 2021
Impact Factor: 5.731

[27]. R. Kumar and U. Kumar, "A Hierarchal cluster framework for wireless sensor network," Int. Conf. Adv. Comput. Commun, IEEE. pp. 46-50, 2012.

[28]. S. H. Chang and T. S. Huang, "A Fuzzy Knowledge Based Fault Tolerance Algorithm in Wireless Sensor Networks," In: 26th Int. Conf. Adv. Inf. Netw. Appl., pp. 891-896, 2012.

[29]. M. R. Brust, D. Turgut, C. H. C. Ribeiro, and M. Kaiser, "Is the clustering coefficient a measure for fault tolerance in wireless sensor networks?," In: IEEE Int. Conf. Commun., pp. 183-187, 2012.

[30]. D. Rong Duh, S. Pei Li, V. Cheng, "Distributed Fault-Tolerant Event Region Detection of Wireless Sensor Networks," Int. J. of Distributed Sensor Networks, pp.1-8, 2013.

[31]. N. Li and J. Hou, "Localized fault-tolerant topology control in wireless ad hoc networks," IEEE Trans. Parallel Distrib. Syst., vol.17, no.4, pp. 307-320, 2006.

[32]. C. Chen, K. Feng Ssu and H. Christine Jiau, "Fault Tolerant Topology Control with Adjustable Transmission Ranges in Wireless Sensor Networks," In: 13th Int. Symposium on Pacific Rim Dependable Computing, IEEE. pp. 131-138, 2007.

[33]. Sitanayah L. Planning the deployment of fault-tolerant wireless sensor networks. PhD, National University of Ireland, Cork, Ireland, 2013.

[34]. Y. Rong-rong, L. Bin, L. Ya-qian and H. Xiao-chen, "Adaptively faulttolerant topology control algorithm for wireless sensor networks," J. of China Universities of Posts and Telecommunications, vol.19, no.2, pp.13-18, 2012.

[35]. Z. Rehena and S. Roy, "Handling Area Fault in Multiple-Sink Wireless Sensor Networks," In: IEEE 3rd Int. Conf. on Advance Computing, IEEE. pp. 458-464, 2013.

[36]. Y. Sun, H. Luo, and S. K. Das, "A trust-based framework for faulttolerant data aggregation in wireless multimedia sensor networks," IEEE Trans. Dependable Secur. Comput., vol.9, no.6, pp.785-797, 2012.

[37]. D. D. Geeta, N. Nalini, and R. C. Biradar, "Fault tolerance in wireless sensor network using hand-off and dynamic power adjustment approach," J. Netw. Comput. Appl., vol.36, no.4, pp.1174-1185, 2013.

[38]. M. Cardei, S. Yang, and J. Wu, "Fault-Tolerant Topology Control for Heterogeneous Wireless Sensor Networks," In: IEEE Int. Conf. Mob. Adhoc Sens. Syst., pp. 1–9, 2007.

[39]. S. Chouikhi, I. El Korbi, Y. Ghamri-Doudane, and L. Azouz Saidane, "A survey on fault tolerance in small and large scale wireless sensor networks," Comput. Commun., pp. 1–16, 2015.

[40]. R. H. Abedi, N. Aslam, and S. Ghani, "Fault tolerance analysis of heterogeneous wireless sensor network," 24th Can. Conf. Electr. Comput. Eng., pp.175-179, 2011.

[41]. Gil Neiger and Sam Toueg, "Automatically increasing the fault-tolerance of distributed systems", in proceedings of the seventh annual ACM Symposium on Principles of distributed computing, pp 248–262, 1988.

[42]. N. Xiong, Y. Yang, M. Cao, J. He and L. Shu, "A Survey on Fault-Tolerance in Distributed Network Systems," 2009 International Conference on Computational Science and Engineering, Vancouver, BC, pp. 1065-1070, 2009.

[43]. Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008