SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V11I4P101

Volume 11 Issue 4, 1-9, April 2024

Review Article

Principles of Fault Tolerance in IT Systems: A Review

Iehab Abduljabbar Kamil[1], Mohanad A. Al-Askari[2]

[1,2]*Department Of Information Systems College of Computer Sciences and Information Technology, University Of Anbar, Iraq.*

[1]*Corresponding Author : iehab.a.k@uoanbar.edu.iq*

Received: 07 February 2024 Revised: 14 March 2 024 Accepted: 02 April 2024 Published: 15 April 2024

Abstract *- This study aims to thoroughly analyse the fundamentals of fault tolerance systems in IT systems. This study aims to give a general understanding of the concepts involved in fault tolerance systems. This research examines fault tolerance concepts in IT systems regarding fault isolation and online repair. Examining fault tolerance in the context of IT systems and going over fault tolerance strategies in fault isolation, online repair, and fault isolation are the key goals of this research topic. A variety of fault-tolerant strategies are applied to increase fault tolerance. Among these, fault isolation is essential to creating a highly available system with fault tolerance. Load balancing and failover techniques are also employed with fault isolation approaches for high availability.*

Keywords *- Fault Tolerance, Availability, Reliability, Scalability, IT Systems.*

1. Introduction

The capacity of a system to function commonly, even if one or more of its elements fails, is known as Fault Tolerance. Fault Tolerance indicates the capability of a system such as a Cloud Cluster, Computer, or Network. Fault Tolerance uses effective techniques to confirm its steady operational activities even in the case of partial failure. Fault Tolerance is based on an effective design strategy that enables systems to function even if components fail to function correctly. Fault Tolerance aids in enhancing the reliability and availability of a system [1]. System downtime and failures are less likely to occur with Fault Tolerance. It can lessen the effects of unforeseen interruptions, software bugs, and hardware malfunctions. Systems featuring Fault Tolerance, for instance, can completely prevent data loss, performance drops and system breakdowns. The ability to withstand mistakes or defects guarantees continuous functioning. By lowering the chance of data loss or corruption, it can aid in the protection of sensitive information. The scope of the research is to explore various factors of Fault Tolerance. Challenges, practical applications, theoretical foundations, and future directions are multiple factors.

2. Fundamentals of Fault Tolerance

2.1. Basic Concepts and Principles

Modern systems design must consider Fault Tolerance, which is especially important for applications where downtime must be kept to a minimum. It is the process of a system functioning correctly even when system faults occur. Six basic concepts and principles enable Fault Tolerance in highly available systems. The six basic concepts and principles of Fault Tolerance are redundancy, fault isolation, fault detection and annunciation, online repair, durability, and dependability. Redundancy is the basic technique in Fault Tolerance systems. It focuses on duplicating crucial resources or elements in the system. Fault isolation is critical to making Fault Tolerance a highly available system [2]. Fault detection and online repair both require inspecting highly available Fault Tolerance systems. Durability is one of the main fundamental concepts of the Fault Tolerance system while maintaining the continuity of the database system. Dependability is a crucial component of trustworthy computing, which entails analysing a system's hostile conditions and creating a plan to keep the system strong despite them.

2.2. Types of Faults in IT Systems (Network, Hardware and Software)

Faults often occur in IT systems, including network, hardware and software. Understanding these kinds of faults in IT systems is necessary for designing a highly effective fault system.

2.2.1. Network Faults

Three different types of faults exist that come under network faults. Link failures, packet loss, latency, and jitter are examples of network faults. When a connection fails, it usually manifests as a sequence of packet losses that might last for several seconds, interspersed with a shift in latency until the link is restored. When a device starts malfunctioning and software issues occur, link failure happens. Router reboots and short maintenance of network equipment can also cause link failure. Packet loss occurs when data packets cannot reach their location on a network [3]. Network congestion is the

main reason for packet loss. Overloaded network devices and discarded packets come under network congestion. User experience and network performance problems can be brought on by packet loss. A packet loss rate of less than 1% or 0.1% is deemed appropriate for many applications.

Nevertheless, reduced packet loss rates could be necessary for specific apps such as VoIP, real-time communication, and online gaming. The time a data packet takes from one location in a network to another is known as latency. Delays that happen while data travels from a computer device's RAM to its CPU can also be referred to by this term [4]. The difference in latency between data packets sent over a network is known as jitter. It also represents the degree to which delay varies between packets.

2.2.2. Hardware Faults

Various hardware components malfunction in the case of component failure. Different hardware components include memory, hard disk drives, CPU, and power supply systems. These components fail to function correctly due to overheating, manufacturing defects, software failure, physical damage, unregulated power supply, viruses, and malware. A hardware malfunction known as a bus error transpires when a process attempts to access memory that the CPU cannot access [5]. The address needs to be corrected for the address bus, which explains this. Programming mistakes and corrupted devices can also cause bus faults in the system: compiler bugs, invalid file descriptors, misaligned data structure, and inadequate memory allocation cause bus errors. Overlocking issues come under hardware faults, which happen when hardware cannot address extra stress. Overlocking issues can cause hardware damage and data corruption. Overlocking problems are responsible for heating issues and corrupted Bios as well.

2.2.3. Software Faults

Errors in software might be more conceptual or logical, and they are blunders committed during development. Bugs are discrete instances of improper behaviour inside a program and express faults in the software's operation [6]. Logical errors occur if the software does not provide any desired output or error message compelling and running a program while responding to the input. A memory leak is a program error and comes under the software faults. When a program fails to release its allocated memory to serve its purpose, it is known as a memory leak. Managing memory allocation inappropriately is the reason for memory leaks.

2.3. Objectives of Fault Tolerance

The objective of fault tolerance is to restrict disruptions from a single point of failure. It confirms business continuity and the high availability of systems. Redundant components, mainly memory disks, are used to accomplish fault tolerance. Potential service outages brought on by software or logical problems are also resolved by fault tolerance.

3. Fault Tolerance Techniques

3.1. Redundancy-based Techniques (Network, Hardware and Software)

Redundancy-based techniques enhance the fault tolerance of applications or software systems. Redundancy comprises replicating essential software systems. If one element fails within the system, other components can continue their functions. Fault Tolerance can be accomplished using redundancy-based techniques for hardware, network, and software systems.

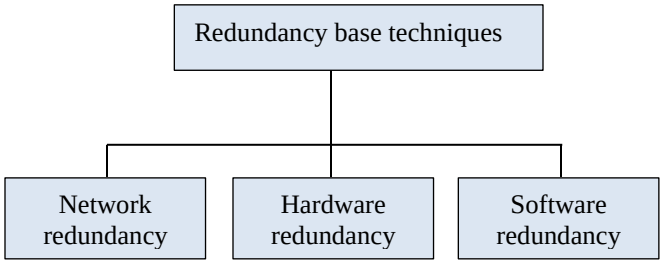

Fig. 1 Three kinds of redundancy-based techniques

3.1.1. Network Redundancy

Installing backup network resources is the network redundancy process, which helps avoid downtime. It entails creating duplicate network infrastructure and executing different instances of essential network services. Several routes for traffic are provided by network redundancy, ensuring that data may continue to flow even in the case of a breakdown [7]. The idea is that devices should be able to switch over if one fails immediately.

3.1.2. Hardware Redundancy

Hardware redundancy adds an identical device or element to a system [8]. When an immediate segment or an appliance dies, this is accomplished to guarantee nil downtime. This sample employs two or more processors instead of a single processor to achieve the identical procedure.

3.1.3. Software Redundancy

One method that can aid in achieving software fault tolerance is Software Redundancy. Replication of software elements, data, or calculations is known as redundancy, and it serves as a backup or alternate option in the event of failures. Redundancy offers system-wide fault tolerance, assisting in making sure that calls are handled by the CSP, even in the event of a software malfunction [9]. An operating system's ability to react to a software malfunction is known as Fault Tolerance. Fault Tolerant systems make use of backup parts that substitute malfunctioning parts automatically.

3.2. Error Detection and Correction Mechanisms (ECC, Checksums and Parity)

Error detection and correction are crucial activities that are effective for exception handling. Writing code to uncover mistakes is known as error detection. The process of controlling and responding to the incidence of specific faults is known as error correction.

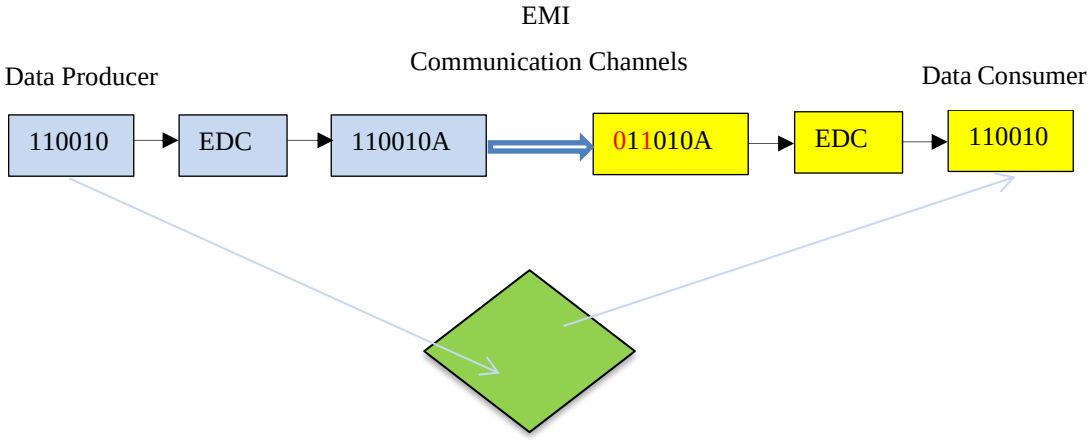

Fig. 2 Error correction code

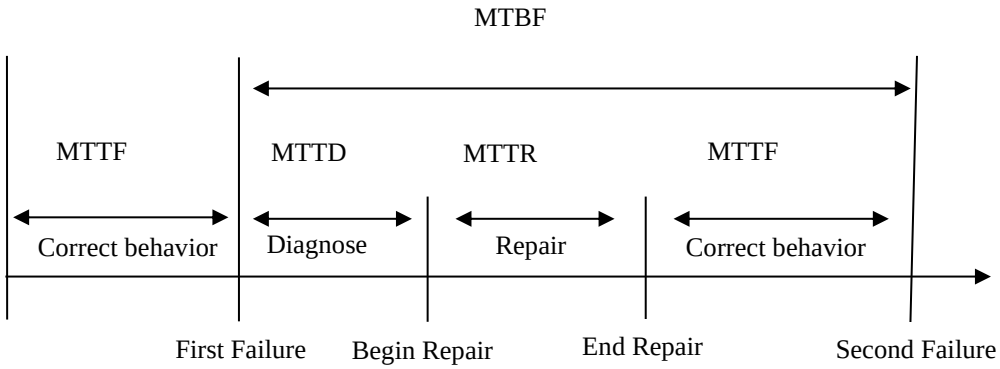

Fig. 3 Evaluation of multiple factors of failure and repair

(Source: [4])

3.2.1. ECC

Error correction code is a strategy for encoding data to define and rectify errors [10]. Adding redundancy to the data enables ECC to serve its work. It helps the decoder discover the message the transmitter has encoded. It is utilized in almost every message transmission scenario, particularly in data storage, where ECCs prevent data corruption.

3.2.2. Checksums

It is known as a unique fingerprint of a file. Checksums are crucial for verifying two identical files. With checksums, the integrity of data files before and after file transfers or backups is easily calculated [11]. The string will alter even if just one erroneous or changed data byte exists.

3.2.3. Parity

Parity is a fundamental error detection method for network communications [12]. A parity bit is an extra 0 or 1 bit attached to the primary original signal. Parity is used to detect errors in the systems. The even and odd parity approaches are available. When using the even parity approach, the bit value is selected to ensure that the transmitted signal's entire amount of 1s, comprising the parity bit, is even.

3.2.4. Mean Time between Failures (MTBF)

This time evaluation provides the investigation details of the expected time to investigate the time differences between two faults. The calculation process introduces two faulty systems which provide or produce faults in the system. The calculation of the MTBF is expressed as follows: MTBF = (NH)/(NF), where NH defines the number of total operational hours, and NF defines the failure numbers.

3.2.5. Mean Time to Failure (MTTF)

This failure evaluation time is calculated with respect to the occurring of the fault in the system. This provides the evaluation between the total operating time, and the use of the total asset values. The calculation of the MTTF is expressed as follows,

MTTF = (TOT)/(TS), where TOT is the total operational time, and TS is the total usable assets.

3.2.6. Mean Time to Repair (MTTR)

This time evaluation is based on the repair value of the overall system. This involves the time for repair and the count of the total number of faults in the system. The calculation of MTTR is expressed as follows,

MTTR = (TTSR)/(NF), where TTSR defines the total time for the repairing of the system, and NF is the failure number.

3.2.7. Availability

The actual time of a machine for evaluating a program is evaluated by calculation the availability of the overall system. This provides the details of the necessary factors which are involved in the calculation of the overall actual time or evaluation. This introduces the evaluation of the failure times of the machine. As per the evaluation, the expression of the availability is as follows,

Availability = (MTBF)/(MTBF+MTTR), this means it involves both mean time between and to failures.

3.3. Failover and Balancing Strategies

Techniques of addressing servers to accomplish high availability are load balancing and failover. Load balancing distributes the burden among several servers to avoid overloading a single one [13]. It enables individuals to run web servers on another server similar to a database server. Switchover: If the primary system fails, transfer the burden to a backup system. One server can take over another server if it fails to perform. Some failover strategies are active-passive failover and hot and cold standby. By using load balancing, request processing is split up across several servers. If the first server contacted is unavailable or operating too slowly, failover routes requests to other servers.

3.4. Recovery Mechanisms (Rollback Recovery, Checkpointing)

Executing the recovery mechanism is quite vital for Fault Tolerance. This enables systems to recover by reducing failures to prevent them from malfunctioning. Rollback recovery and checkpointing are the two most common recovery mechanisms.

3.4.1. Rollback Recovery

A distributed system can be brought back to a consistent state through rollback recovery in the event of a failure [14]. It's only one method among several used to improve distributed systems' availability and dependability. A two-phase commit technique is comparable to the rollback recovery algorithm.

3.4.2. Checkpointing

A method called checkpointing makes a copy of an application's current state so that, if it malfunctions, it may be restored and run again later. Checkpointing is vital when managing protracted agendas on computer approaches that tend to oversight [15].

Furthermore, it operates in ingrained procedures to expand steadfastness by occasionally reserving the system's circumstances.

3.5. Diversity Techniques (Algorithmic Diversity & Data Diversity)

Diversity techniques are used in Fault Tolerance to increase the system's resilience while introducing variations in the system. The two most helpful diversity techniques are algorithmic diversity and data diversity.

3.5.1. Algorithmic Diversity

The main aim of algorithmic diversity is to utilize two or multiple algorithms to fulfil the same function. Algorithmic diversity helps improve Fault Tolerance by minimizing the impact level of different kinds of system failures [16].

3.5.2. Data Diversity

Data Diversity emphasizes introducing variation in input parameters to diversify variation in output [17]. This approach to data diversity enhances Fault Tolerance by making it capable of detecting and dealing with fault sequences.

4. Design Considerations for Fault Tolerance

4.1. Reliability Modelling and Analysis

Reliability metrics estimate the capacity and efficiency of a system for performing its functions without any malfunctions over a certain period.

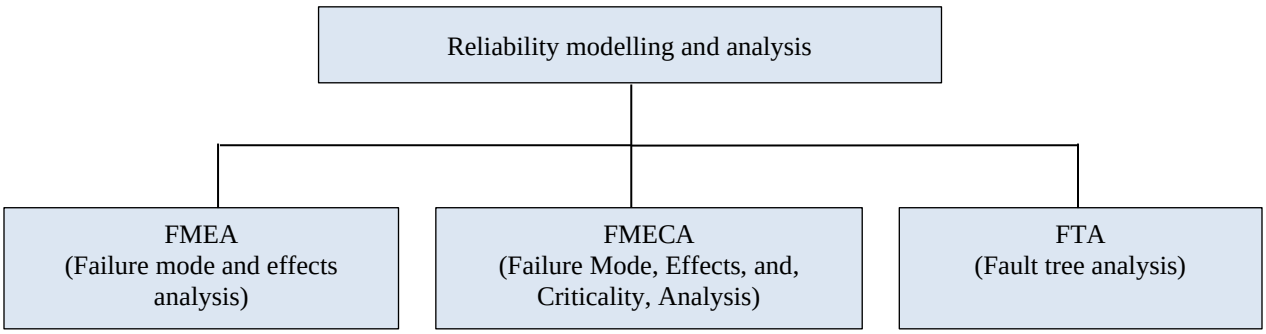

Fig. 4 Three analyses of reliability modelling

4.1.1. FMEA (Failure Mode and Effects Analysis)

A systematic, inductive, bottom-up approach to failure mode identification and prioritisation, FMEA can be carried out at the piece-part or functional levels.

4.1.2. FMECA (Failure Mode, Effects, and Criticality Analysis)

An expansion of FMEA with a criticality analysis included. FMECA takes into account all potential system failure modes before providing a metrics-based method for estimating the likelihood of each failure [18]. Organizations may prioritize which topics to focus on with the aid of FMECA.

4.1.3. FTA (Fault Tree Analysis)

Employs a top-down logical method to determine the probability that an undesirable event will occur.

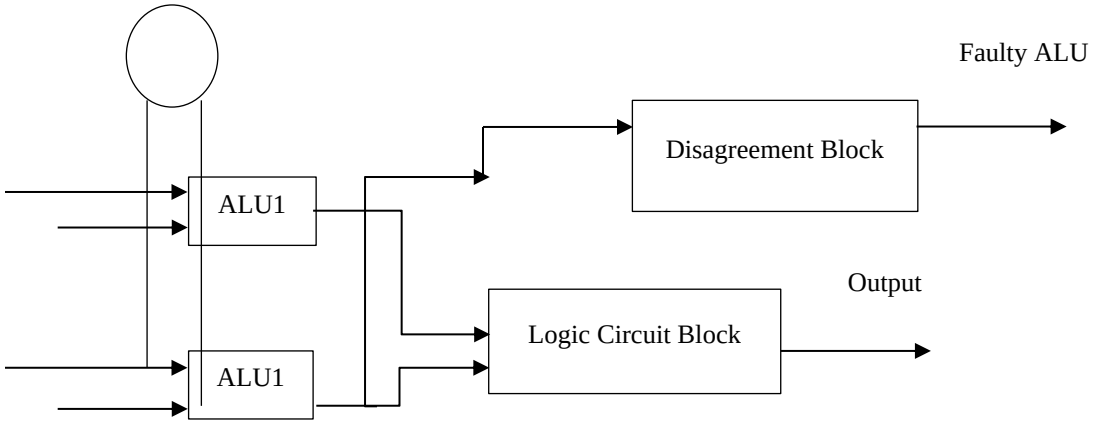

Fig. 5 Block diagram of fault tolerance technique

(Source: Self-developed)

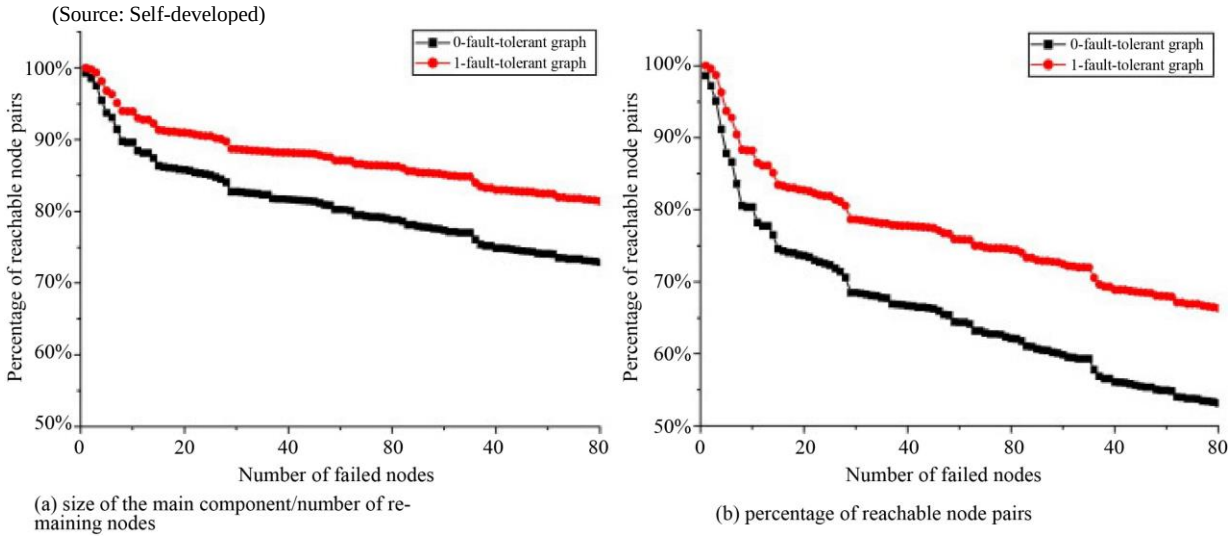

Fig. 6 Fault-tolerance system plot

(Source: [10])

The design process of the fault-tolerance technique of the computer system is demonstrated by the above block diagram. This provides the details of the fault tolerance facility in a computer. The process involves an input section which is connected with the ALU units of the computer. This also provides two different outcomes in which one produces the faulty ALU, whereas another one produces a proper outcome. The proper outcome is collected from the logical circuit block of the computer.

4.1.4. Fault Tolerance Architecture (Replication-based, N-Version Programming)

Fault Tolerance architecture indicates designing the architecture for the systems to ensure its reliability and availability before failure.

4.1.5. Replication-based Architecture

Based on a centralized architecture, this design ensures that duplicate servers are integrated into the system to provide an alternative for a faulty component or system. This

architecture enhances the resiliency of the system in case of Fault Tolerance.

4.1.6. N-version Programming

The NVP structure approach aims at Software Fault Tolerance. According to this concept, the software can still run and give correct results even if there are errors. To improve systems that are essential for safety reasons, NVP is used to increase software quality [19]. Therefore, any single scar does not certify the malfunctioning of any module's functionality because, instead, the defects may have been localized to a singular interpretation among several arrangements of the NVP software module.

The fault-tolerance system plot provides two types of plots which one provides the size of the main section and the number of remaining nodes, and the other one defines the percentage of reachable node pairs. This provides the evaluation of the data functionality, which assists in finding out the 0 and 1 fault-tolerance plots.

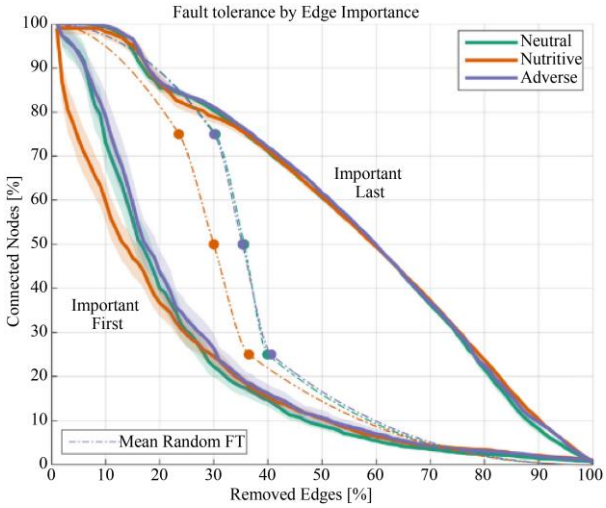

Fig. 7 Fault tolerance by edge importance

(Source: [12])

The fault tolerance process provides the graphical representation of the edge importance process. This process introduces mean random FT, which is highlighted in the main plot. The plot defines the relation between removed edges and the connected nodes. The plot also provides various functional factors such as neutral, adverse, and nutritive.

4.2. Scalability and Performance Implications

These factors are vital to consider when designing an efficient fault tolerance system. These two elements allow designers to enhance dependability and effectiveness in IT systems. Replication expands the system's capacity by creating more clones. It can also lead to scalability if data is spread among various nodes. Since replicated data is available across numerous places, it may be retrieved more rapidly. It can enhance query performance and lower network latency. N-version programming is a helpful technique to raise software quality, particularly for safety-critical systems [20]. It is anticipated that this would improve software availability.

5. Challenges in Fault Tolerance

5.1. The Complexity of the Modern IT System

IT systems constantly change over time, and various organizations have taken the initiative to adopt necessary IT systems. Developing fault tolerance to detect and mitigate the failure impact over any system requires complex algorithms, architecture, and protocols.

5.2. Trade-offs between Fault Tolerance and Other System Attributes

The capacity of a system to function continuously, even if many components fail, is known as fault tolerance. The additional hardware required for embedded Fault Tolerance drives up the system's expense. Redundancy, or the availability of backup parts or different paths to take over in the case of a breakdown, is a feature that fault-tolerant networks frequently incorporate. The network's overall performance and stability are the main concerns of reliability. From a security perspective, a reliable system must be able to ward against evil attempts. According to the Fault Tolerance perspective, a reliable system cannot rely on any one part performing as intended [21]. The capacity of a system to function continuously with little chance of failure is known as high availability. A plan for continuity of operations will incorporate high availability and fault tolerance.

5.3. Dynamic Environments and Changing System Requirements

Dynamic environments and changing system requirements constantly challenge building Fault Tolerance in an IT system. These factors bring complexities and uncertainties in designing Fault Tolerance for the systems to deal with failure.

5.4. Integration with Emerging Technologies (Cloud Computing and IoT)

Embracing these technologies requires addressing challenges from the end of modern businesses to acquire benefits. Integration with IoT and Cloud Computing is helpful in the case of Fault Tolerance. These technologies improve the performance and reliability of the systems. Various businesses face security, data storage, and interoperability challenges in integrating emerging technologies with their systems [22]. Data privacy, performance, cost management, scalability, data loss, and recovery are other challenges.

6. Case Studies & Applications

6.1. Real-world Examples of Fault Tolerance Implementation in IT Systems

The capacity of a system to withstand mistakes and disruptions without losing functioning is known as fault

tolerance. Here are a few instances of IT systems that exhibit fault tolerance:

6.1.1. Servers for Backups

Another server with the same configuration can take over in the event of a failure.

6.1.2. Supplementary CPUs

Redundant processors in a fault-tolerant computer system can carry out the same commands at the same time.

6.1.3. RAID

A redundant array of cheap discs (RAID) improves performance by combining the physical components of discs [23].

6.1.4. Identification and Reversal of Course

Every time a calculation is made, the system is tested using this method. When there is data corruption or processing breakdown, it is helpful.

6.1.5. Self-Healing

Devices are capable of self-healing features, which not only continue when a mistake happens but also eventually automatically fix the issue.

7. Future Direction

7.1. Emerging Trends and Technologies in Fault Tolerance

Some emerging technologies in the context of fault tolerance are Artificial Intelligence (AI), Blockchain, Machine Learning, and Quantum Computing. These technologies heavily impact Fault Tolerance [24]. Other technologies in the case of Fault Tolerance are genomics, sustainable energy, large-scale scientific information, and quantum information science.

7.2. Research Challenges and Opportunities

7.2.1. Fault Tolerance in Distributed Systems

Researching Fault Tolerance in simultaneous and distributed grid settings is a challenging problem [25]. This results from the lengthy processing times associated with computer-intensive grid applications.

7.2.2. Load Balancing

Researchers find load balancing to be a challenging endeavour. Multiple path-based routing and load-balancing techniques are used in WSNs to evenly divide traffic among several CHs or accessible links/paths.

7.2.3. Data Replication

Big IoT-generated data presents several processing issues. Replicating data to increase Fault Tolerance, dependability, and accessibility is one of the trickiest issues.

7.2.4. Fault Recovery

An irregular execution time results from a single task failure, which hurts all the other data processing jobs that are operating normally [26].

The concept of the structural formation of the fault tolerance system is demonstrated in the above portion. This provides the introduction of the web application with the load balancer. This load balancer is used to control various datacentres which are used to store the data. The failover section is used to connect with the standby server, which is introduced when a fault has appeared in the system.

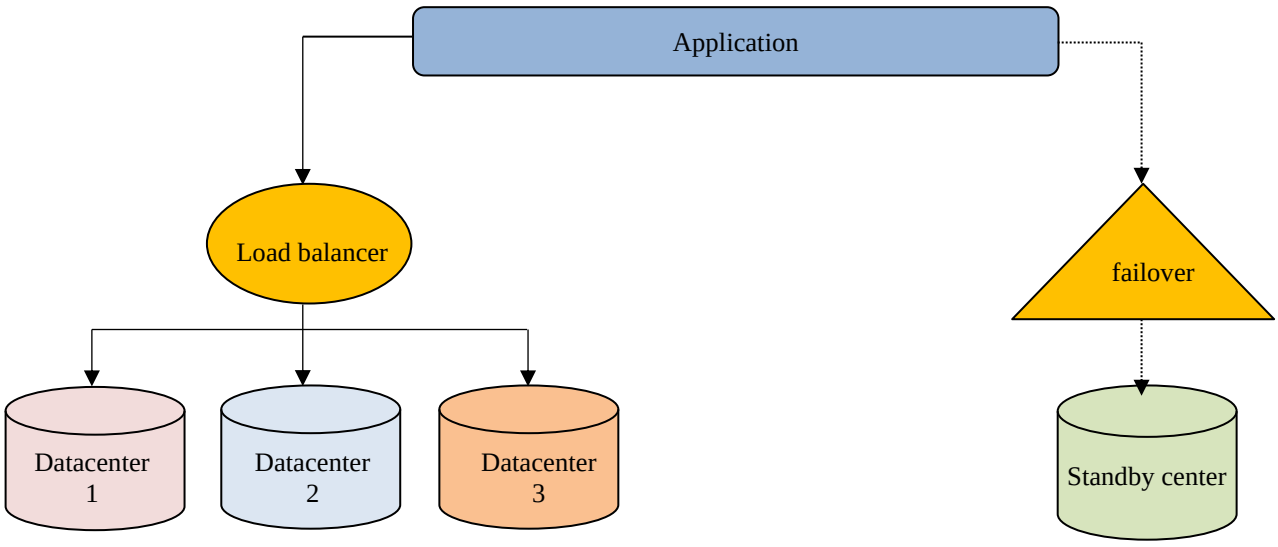

Fig. 8 Fault-tolerance system design structure

7.3. Potential Areas for Innovation and Improvement

Businesses must focus on improving the quality of their offerings while embracing necessary changes to drive innovation. Adopting emerging technologies to meet market demands and clients' perspectives is another area for improvement in the case of fault tolerance [27].

Improvement of products, processes, and services would allow businesses to drive product, process, and service innovation.

8. Conclusion

A system's ability to continue operating normally if one or more of its components fail is known as fault tolerance. It reduces the impact of errors and disruptions, improving system availability and dependability. Fault Tolerance is achieved by employing redundancy-based approaches, error detection and correction systems, failover and balancing strategies, recovery mechanisms, and diversity techniques. Reliability modelling and analysis, such as FMEA and FMECA, are crucial when building Fault Tolerant systems.

References

[1] Shadi Attarha et al., "Virtualization Management Concept for Flexible and Fault-Tolerant Smart Grid Service Provision," *Energies*, vol. 13, no. 9, pp. 1-16, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[2] Han Bao, Tate Shorthill, and Hongbin Zhang, "Hazard Analysis for Identifying Common Cause Failures of Digital Safety Systems Using a Redundancy-Guided Systems-Theoretic Approach," *Annals of Nuclear Energy*, vol. 148, pp. 1-22, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Tobias Distler, "Byzantine Fault-Tolerant State-Machine Replication from a Systems Perspective," *ACM Computing Surveys*, vol. 54, no. 1, pp. 1-38, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[4] Maria Bonaventura Forleo et al., "Analysing the Efficiency of Diversified Farms: Evidences From Italian FADN Data," *Journal of Rural Studies*, vol. 82, pp. 262-270, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Ionel Gog, Michael Isard, and Martín Abadi, "Falkirk Wheel: Rollback Recovery for Dataflow Systems," *Proceedings of the ACM Symposium on Cloud Computing*, pp. 373-387, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Geddam Kiran Kumar, and Devaraj Elangovan, "Review on Fault-Diagnosis and Fault-Tolerance for DC–DC Converters," *IET Power Electronics*, vol. 13, no. 1, pp. 1-13, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Rakesh Kumar et al., "The Mystery of the Failing Jobs: Insights from Operational Data from Two University-Wide Computing Systems," *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Valencia, Spain, pp. 158-171, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[8] Priti Kumari, and Parmeet Kaur, "A Survey of Fault Tolerance in Cloud Computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 33, no. 10, pp. 1159-1176, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[9] P. Dhivya Lakshmi, "Constructing Low-Density Parity-Check Codes in Digital Communication System," *ICTACT Journal on Communication Technology*, vol. 11, no. 2, pp. 2198-2202, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Bastien Lecoeur, Hasan Mohsin, and Alastair F. Donaldson, "Program Reconditioning: Avoiding Undefined Behaviour When Finding and Reducing Compiler Bugs," *Proceedings of the ACM on Programming Languages*, vol. 7, pp. 1801-1825, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Richard Li, Kiran Makhijani, and Lijun Dong, "New IP: A Data Packet Framework to Evolve the Internet," *IEEE 21st International Conference on High Performance Switching and Routing*, pp. 1-8, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[12] Ján Mach, Lukáš Kohútka, and Pavel Čičák, "On-Chip Bus Protection against Soft Errors," *Electronics*, vol. 12, no. 22, pp. 1-16, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[13] Omid Maghazei, Michael A. Lewis, and Torbjørn H. Netland, "Emerging Technologies and the Use Case: A Multi-Year Study of Drone Adoption," *Journal of Operations Management*, vol. 68, no. 6-7, pp. 560-591, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Mohammad Mahdavi, and Ziawasch Abedjan, "Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 1948-1961, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Júlio Mendonça, Fumio Machida, and Marcus Völp, "Enhancing the Reliability of Perception Systems using N-version Programming and Rejuvenation," *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Porto, Portugal, pp. 149-156, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Tayyab Muhammad et al., "AOptimizing Network Paths: In-Depth Analysis and Insights on Segment Routing," *Journal of Data Acquisition and Processing*, vol. 38, no. 4, pp. 1942-1963, 2023. [Google Scholar] [Publisher Link]

[17] Prakai Nadee, and Preecha Somwang, "Efficient Incremental Data Backup of Unison Synchronize Approach," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 5, pp. 2707-2715, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] S. Sai Haree Ram, S. Uppala Durga Samrith, and V. Pandimurugan, "Decentralized Cloud Disaster Recovery using Consensus Algorithm," *2nd International Conference on Automation, Computing and Renewable Systems*, Pudukkottai, India, pp. 923-930, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[19] Sepideh Safari et al., "A Survey of Fault-Tolerance Techniques for Embedded Systems from the Perspective of Power, Energy, and Thermal Issues," *IEEE Access*, vol. 10, pp. 12229-12251, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[20] Zhibing Sha et al., "Proactive Stripe Reconstruction to Improve Cache Use Efficiency of SSD-Based RAID Systems," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1-18, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[21] Nadir Subasi, Ufuk Guner, and Ilker Ustoglu, "N-Version Programming Approach with Implicit Safety Guarantee for Complex Dynamic System Stabilization Applications," *Measurement and Control*, vol. 54, no. 3-4, pp. 269-278, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] Ola Hani Fathi Sultan, and Turkan Ahmed Khaleel, "Challenges of Load Balancing Techniques in Cloud Environment: A Review," *Al-Rafidain Engineering Journal*, vol. 27, no. 2, pp. 227-235, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[23] Pejman Memar, "Simulation Setup for Investigating the Error Detection and Correction Codes Effectiveness Under Harsh Electromagnetic Interference," *Pan European Training Research and Education Network on Electromagnetic Risk Management*, 2020. [Publisher Link]

[24] Rongxi Wang et al., "Reliability Analysis of Complex Electromechanical Systems: State of the Art, Challenges, and Prospects," *Quality and Reliability Engineering International*, vol. 38, no. 7, pp. 3935-3969, 2020. [Cross Ref] [Google Scholar] [Publisher Link]

[25] Zhenyu Xu et al., "A Bus Authentication and Anti-Probing Architecture Extending Hardware Trusted Computing Base Off CPU Chips and Beyond," *ACM/IEEE 47th Annual International Symposium on Computer Architecture*, Valencia, Spain, pp. 749-761, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[26] Guojie Yang et al., "Interoperability and Data Storage in Internet of Multimedia Things: Investigating Current Trends, Research Challenges and Future Directions," *IEEE Access*, vol. 8, pp.124382-124401, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[27] Yi Zhang et al., "Dynamic Job Shop Scheduling Based on Deep Reinforcement Learning for Multi-Agent Manufacturing Systems," *Robotics and Computer-Integrated Manufacturing*, vol. 78, 2022. [CrossRef] [Google Scholar] [Publisher Link]