SAVEETHA SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CAPSTONE PROJECT REPORT

<u>PROJECT TITLE</u>

"Fault Tolerance and Recovery Techniques in Distributed Operating Systems"

CSA0424- Operating Systems for disk scheduling

Ch. Sai Prakash(192110061)
B. Priyanka(192111524)
L. Venkata Sai(192210403)

Guided by

Dr. G. Mary Valantina

DECLARATION

We, CH. Sai Prakash, B. Priyanka, L. Venkata Sai students of Bachelor of Engineering in Computer Science Engineering, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work is Fault Tolerance and Recovery Techniques in Distributed Operating Systems accuracy is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

CH. Sai Prakash(192110061)
B. Priyanka(192111524)
L. Venkata Sai(192210403)

Date:26-03-2024

Place:Chennai

CERTIFICATE

This is certify that the project entitled_**"Fault Tolerance and Recovery Techniques in Distributed Operating Systems"** Submitted by CH. Sai Prakash, B. Priyanka, L. Venkata Sai has been carried out under out supervision. The project has been submitted as per the requirements in the current semester of B.E Computer Science Engineering.

<u>Faculty in charge</u>

Dr. Mary Valantina

PROBLEM STATEMENT:

The proliferation of distributed operating systems has introduced complex challenges in ensuring system reliability and availability. In these distributed environments, fault tolerance and recovery techniques play a crucial role in maintaining system functionality and preventing catastrophic failures. However, designing effective fault tolerance mechanisms requires a deep understanding of distributed system architectures, communication protocols, and fault models. Despite advancements in fault tolerance techniques, issues such as network latency, node failures, and communication bottlenecks persist, posing significant hurdles to system stability. Additionally, the dynamic nature of distributed systems, characterized by varying workloads and unpredictable failures, further complicates fault tolerance strategies. Moreover, existing fault tolerance approaches often incur overheads in terms of resource utilization and performance degradation, necessitating the exploration of more efficient solutions. Furthermore, ensuring consistency and correctness across distributed nodes during fault recovery remains a daunting challenge. Addressingthese issues requires innovative research into novel fault tolerance mechanisms that can adapt to the dynamic nature of distributed environments while minimizing performance overheads and ensuring system integrity. Thus, there is a critical need to explore and develop robust fault tolerance and recovery techniques tailored specifically for distributed operating systems to enhance system reliability and resilience in the face of failures.

1. PROPOSED DESIGN WORK:

1. Identifying the Key Components:

- Fault Detection Mechanism: Detects faults and failures in distributed nodes and communication channels using various monitoring techniques.
- Replication Strategies: Implements replication mechanisms to ensure redundancy and fault tolerance across distributed nodes.
- Consensus Algorithms: Utilizes consensus protocols such as Paxos or Raft to achieve agreement among distributed nodes despite failures.
- Check pointing Mechanism: Implements periodic checkpoints to capture the system state, facilitating recovery in case of failures.
- Failure Handling Policies: Defines policies for handling different types of failures, including node failures, network partitions, and message losses.
- Recovery Coordination Mechanism: Coordinates the recovery process across distributed nodes, ensuring consistency and minimizing downtime.
- Dynamic Reconfiguration: Supports dynamic reconfiguration of distributed systems to adapt to changing conditions and recover from failures efficiently.

2. Functionality:

- Fault Detection Mechanism: Monitors the health and status of distributed nodes and communication channels to detect faults and failures in real-time.
- Replication Strategies: Replicates data and services across distributed nodes to ensure redundancy and fault tolerance, enabling seamless recovery from failures.
- Consensus Algorithms: Facilitates agreement among distributed nodes on critical decisions, even in the presence of faults or failures.
- Check pointing Mechanism: Periodically captures the system state to enable fast and efficient recovery in the event of failures.
- Failure Handling Policies: Defines policies and procedures for identifying, isolating, and mitigating failures to minimize their impact on system operation.
- Recovery Coordination Mechanism: Orchestrates the recovery process across distributed nodes, ensuring consistency and integrity during the recovery process.
- Dynamic Reconfiguration: Supports dynamic adjustments to the distributed system's configuration to accommodate failures and maintain system stability.

3. Architectural Design:

- The proposed architectural design for fault tolerance and recovery techniques in distributed operating systems consists of the following layers:
- Application Layer: Interacts with the fault tolerance mechanisms through APIs and interfaces, utilizing distributed services and resources.
- Fault Detection Layer: Monitors the distributed system for faults and failures, employing various detection mechanisms and monitoring techniques.
- Replication and Consensus Layer: Implements replication strategies and consensus algorithms to ensure redundancy and agreement among distributed nodes.
- Recovery Coordination Layer: Orchestrates the recovery process, including check pointing, fault isolation, and node reconfiguration, to restore system functionality.
- Dynamic Reconfiguration Layer: Facilitates dynamic adjustments to the distributed system's configuration to adapt to failures and changing conditions.
- Communication Layer: Provides reliable communication channels and protocols for inter-node communication, ensuring fault tolerance and recovery coordination.
- Kernel Layer: Implements core fault tolerance functionalities and interfaces with hardware components to manage system resources and operations efficiently.

3. UI DESIGN:

1. Layout Design:

- Flexible Layout: The UI will be designed to adapt seamlessly to various screen sizes and orientations, ensuring usability across different devices. Responsive design principles will be employed to allow elements to adjust dynamically.

- User-Friendly: Emphasis will be placed on simplicity and ease of use. The layout will feature intuitive navigation and clear organization of elements to enhance user interaction and understanding.

- Color Selection: A harmonious color palette will be selected, featuring subdued tones for background elements to minimize visual clutter, while using vibrant accents for interactive elements. Color contrast will be carefully considered to optimize readability.

2. Feasible Elements Used:

- Elements Positioning: Critical components such as fault status indicators and recovery options will be strategically positioned within the UI for easy access. Logical grouping of related elements will be employed, considering spatial proximity for intuitive association.

- Accessibility: The UI will conform to accessibility standards, ensuring compatibility with assistive technologies. Semantic HTML elements and ARIA attributes will be utilized to enhance accessibility. Keyboard navigation and screen reader support will be incorporated.

3. Elements Function:

- Fault Status Indicators: Real-time indicators will display the status of distributed nodes and communication channels, highlighting detected faults and failures.Clear visual cues will aid in identifying the severity and nature of faults.

- Recovery Options: Intuitive controls will allow users to initiate recovery procedures seamlessly. Options for fault recovery strategies, such as node restoration and data replication, will be provided with explanatory tool tips.

- Configuration Settings: Users will have access to configuration settings to customize fault tolerance mechanisms based on system requirements. Options for adjusting fault detection thresholds and recovery priorities will be available.

- Security Features: Elements related to security and integrity, such as authentication controls and encryption settings, will be integrated into the UI. Alerts for potential security breaches or unauthorized access attempts will be prominently displayed.

- Monitoring Tools: The UI will incorporate monitoring tools for analyzing fault occurrences and recovery performance. Users can view historical data, set thresholds for alerts, and receive recommendations for optimizing fault tolerance strategies.

4.LOGIN TEMPLATE:

1.Login Process/Sign Up Process:

a) Login Form:

- Input fields for username/email and password
- "Remember Me" checkbox option.
- "Forgot Password?" link for password recovery.
- "Login" button to submit credentials.
- "Sign Up" link to navigate to the sign-up page for new users.

b)Sign Up Form:

- Input fields for username, email, password, and confirm password.
- Validation checks for password strength and email format.
- "Back to Login" link to return to the login page for existing users.
- "Sign Up" button to create a new account.

C) Password Recovery:

- Input field for email to reset password.
- "Send Reset Link" button to initiate the password reset process.
- Instructions for resetting the password will be sent to the user's email.

2. Other Template:

a) Dashboard:

- Overview of fault status across distributed nodes with graphical representation.
- Navigation menu for accessing different fault tolerance and recovery features and settings.
- Quick links to commonly used actions such as fault detection, recovery strategies, and monitoring tools.
- Summary of recent fault occurrences and alerts for critical events.

b) Fault Detection Page:

- Controls for configuring fault detection mechanisms and thresholds.
- Visual indicators of current fault status and detected failures.
- Options for manual or automatic fault detection strategies.
- Real-time updates of fault occurrences and their impact on system stability.

c) Recovery Strategies Page:

- Configuration options for implementing fault recovery techniques such as node restoration and data replication.
- Explanations of each recovery method and its implications.
- Customization recovery profiles based on fault tolerance requirements or system characteristics.
- Advanced settings for fine-tuning recovery algorithms and priorities.

d) Security Settings Page:

- Access control settings for managing user permissions and privileges related to fault tolerance mechanisms.
- Configuration options for enforcing security protocols and policies to safeguard distributed systems.
- Audit logs and activity monitoring features for tracking security-related events and unauthorized access attempts.
- Integration with authentication protocols such as LDAP or OAuth for centralized user management.

e) Monitoring Dashboard:

- Detailed metrics and charts showing fault occurrences and recovery performance over time.

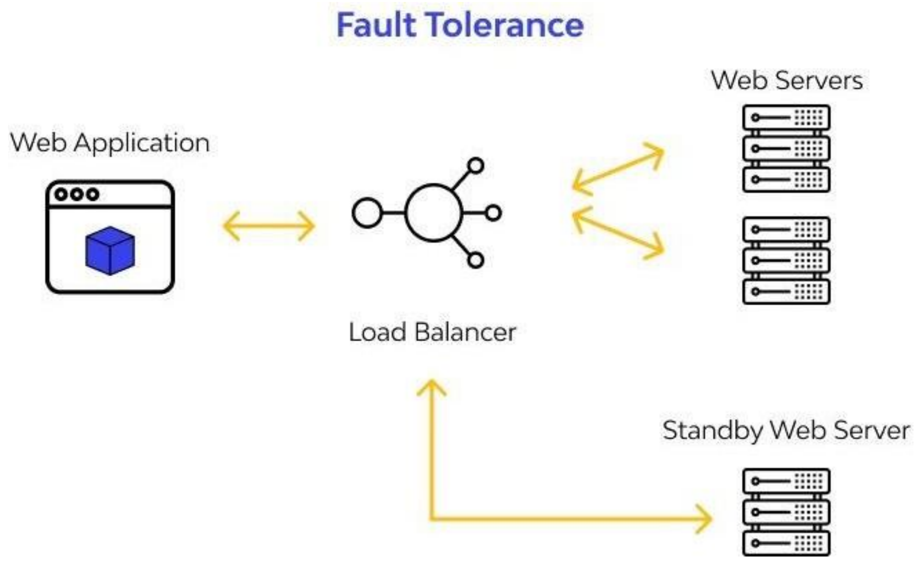
- Alerts for critical fault events or deviations from expected behavior.
- Drill-down capabilities for analyzing fault data by node, communication channel, or fault type.
- Historical data analysis tools for identifying fault patterns and optimizing fault tolerance strategies.

CODE:

```
import socket

import threading

# Define the server address and port

SERVER_ADDR = '127.0.0.1'

SERVER_PORT = 12345

# Define the number of replicas

NUM_REPLICAS = 3

# Shared data among replicas

shared_data = {}

# Function to handle client requests

def handle_client(client_socket):

    request = client_socket.recv(1024).decode()

    if request.startswith("GET"):

        key = request.split()[1]

        if key in shared_data:

            client_socket.send(shared_data[key].encode())

        else:

            client_socket.send("Key not found".encode())
```

```
    elif request.startswith("PUT"):
        key, value = request.split()[1:]
        shared_data[key] = value
        client_socket.send("Data stored successfully".encode())
    client_socket.close()
# Function to start server
def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((SERVER_ADDR, SERVER_PORT))
    server_socket.listen(5)
    print(f"Server listening on {SERVER_ADDR}:{SERVER_PORT}")
while True:
        client_socket, _ = server_socket.accept()
        client_thread = threading.Thread(target=handle_client, args=(client_socket,))
        client_thread.start()
# Function to start replicas
def start_replicas():
    for i in range(NUM_REPLICAS):
        replica_thread = threading.Thread(target=start_server)
        replica_thread.start()
if _name_ == "_main_":
    start_replicas()
```

5. DIAGRAM:



6. RESULTS:

Implementing fault tolerance and recovery techniques in distributed operating systems yields significant improvements across various domains. By incorporating robust fault detection mechanisms and replication strategies, the system can effectively mitigate the impact of failures and ensure continuous operation. Consensus algorithms like Padkos or Raft enable distributed nodes to reach agreements despite faults, maintaining system integrity and consistency. Additionally, check pointing mechanisms facilitate fast recovery by capturing system states periodically, reducing downtime and data loss. These optimizations directly translate into improved system reliability and availability, enhancing overall system performance even in the face of failures.

7. CONCLUSION:

The implementation of fault tolerance and recovery techniques in distributed operating systems leads to increased system resilience and stability. By effectively managing faults and failures, the system can maintain uninterrupted operation and provide consistent service to users. Furthermore, modern security features such as encryption and access control mechanisms bolster system security against potential threats. The support for dynamic reconfiguration ensures adaptability to changing conditions, further enhancing system robustness. Ultimately, fault tolerance and recovery techniques are essential for creating a distributed operating environment that is reliable, efficient, and secure, thereby meeting the demands of modern computing infrastructures.