

Communication induced checkpointing based fault tolerance mechanism using deep-learning in IoT applications

Sowjanya Lakshmi A¹, Vanipriya Ch²

¹Department of Computer Science and Engineering, Sir M. Visvesvaraya Institute of Technology, Bengaluru, India

²Department of Master of Computer Application, Sir M. Visvesvaraya Institute of Technology, Bengaluru, India

Article Info

Article history:

Received Jun 11, 2024

Revised Oct 5, 2024

Accepted Oct 9, 2024

Keywords:

Checkpoint at intermediate node

Communication induced checkpointing based fault tolerance mechanism

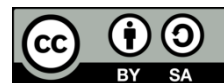
Long short-term memory algorithm

Transient faults

ABSTRACT

Internet of things (IoT) is increasingly used in diverse environments such as healthcare, industry and agriculture. They carry a risk of adverse effects if they make decisions based on faulty information. Software faults, especially transient faults are a primary contributor to deficient decision-making. The existing fault tolerant mechanisms often suffer from checkpoint overheads as checkpoints are placed in all the nodes. This paper describes a novel communication induced checkpointing based fault tolerance mechanism (CIC-FTM) designed to efficiently recover from transient faults, while minimizing useless and forced checkpoints. Long short-term memory (LSTM) based deep learning algorithm is used in our approach to predict fault occurrences and strategically place checkpoints. The proposed method also in turn improve system reliability and performance. Experimental results demonstrate the effectiveness of proposed CIC-FTM in IoT environment by minimizing the practicable operating time for checkpointing and back propagation, compared to traditional fault-tolerance mechanisms.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Sowjanya Lakshmi A

Department of Computer Science and Engineering, Sir. M. Visvesvaraya Institute of Technology

Bengaluru -562157, India

Email: sowjanya.engg@gmail.com

1. INTRODUCTION

The internet of things (IoT) applications are pervasive in various domains such as smart homes, industries, agriculture and healthcare. These applications rely heavily on the integrity of a data transactions among interconnected devices to function correctly [1]. Interactions occur through message exchanges, with operations and decisions reliant on data transmitted and received across IoT nodes [2], [3]. Sensors collect unstructured data, later converted to structured input for IoT processing units. Processors, linked via standard networks, exchange messages through dedicated channels to minimize communication delays [4]-[7]. Fog computing, positioning computational devices closer to IoT devices, aids time-critical applications. Data processing occurs in fog computing instead of cloud computing for swift decision-making. The architecture diagram, Figure 1 for agriculture IoT depicts this setup. IoT systems are prone to transient faults during data transactions, which can lead to incorrect operations and potential system damage. Software transient faults arise from issues like data transactions, server failures and breaches [8]-[10] can disrupt IoT systems but are typically recoverable by restarting tasks. Transient faults like sequence number, checksum, null character and out-of-range errors that contribute to risks [7], [10]-[13] are detected in this paper's first contribution.

Various checkpointing based fault tolerance mechanisms (FTM) are effectively recovering faults, but not addressing transient faults and also recovery is at the cost of checkpointing overheads and storage overheads. The following observations are found in the literature review-various successful FTMs utilize

rollback via checkpointing, categorized as un-coordinated, coordinated and communication induced checkpointing (CIC) based FTMs [6], [10], [14]-[21]. Checkpointing saves crucial information temporarily, including code, data, status, register contents, environment conditions, access counts of the files, file pointers and file related details, ensuring consistent system state [10], [22]. In un-coordinated checkpointing FTMs, where checkpoints are set at fixed intervals or strategically, independent of processor operations, causing inefficiency due to inconsistent nodes and the Domino effect [8], [10], [11], [23]-[25]. Coordinated checkpointing synchronizes processes through system messages to ensure a consistent global state, albeit with storage overheads [8], [10]. CIC combines aspects of both techniques, placing checkpoints based on application messages to avoid unnecessary ones but risking missing useful ones, necessitating forced checkpoints placements for fault recovery [5], [7], [9], [25]. CIC allows designers to place checkpoints based on specific conditions conveyed by explicit messages [8], [18], [20], [23]-[29]. Widely used in various applications including parallel and distributed computing, CIC-FTMs are under constant research, especially in IoT.

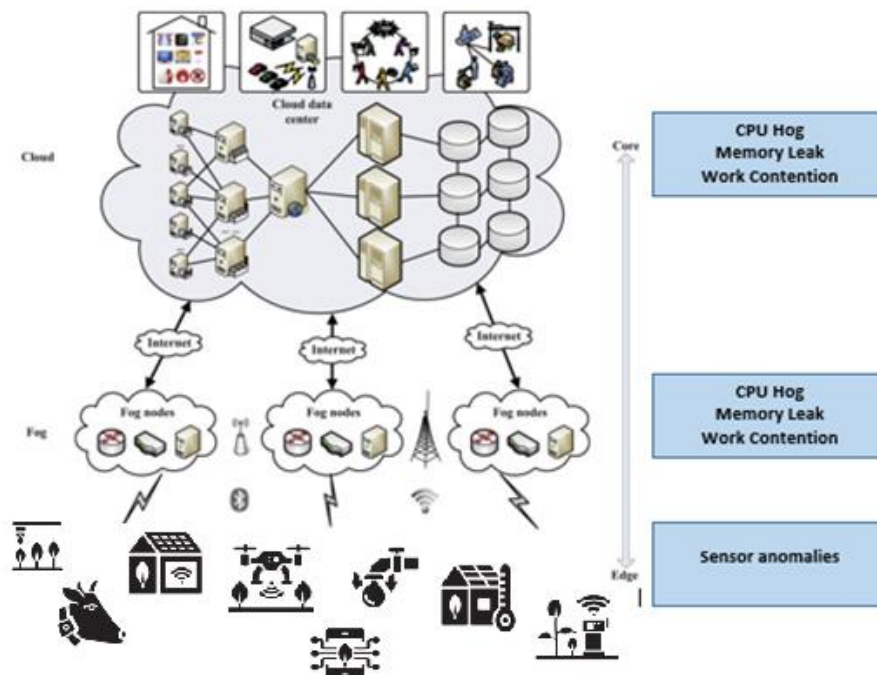


Figure 1. Fog enabled cloud infrastructure

Eles *et al.* [24] integrates error detection and equidistant checkpointing with rollback recovery and active replication to handle faults into software architecture, aiming to meet performance and cost constraints in safety-critical applications. This approach assumes a limited number of faults and leads to potential time overheads even though it primarily handles transient faults. Helary *et al.* [25] focuses on optimizing the checkpointing mechanism in distributed systems by identifying and eliminating redundant checkpoints through a communication-based FTM approach. This method has increased complexity and difficulties in dynamic environments. The continued research [26] focus on implementation of CIC techniques to determine consistent snapshots efficiently by initiating checkpoints through marker messages, logging messages in transit and ensuring consistency through regular monitoring. Hence checkpoint overheads are reduced and the approach is adaptable to both centralized and decentralized systems but, still faces complexity issues in implementation to furnish the constraints and not effectively handling complex failure scenarios. Garcia *et al.* [22], [30], and Vieira *et al.* [31] explores an advanced checkpointing technique designed to enhance fault tolerance in distributed systems using local checkpoints and recovery lines, where each process independently manages its checkpoints based on local knowledge and coordination is used to establish consistent global recovery lines. This approach reduce checkpointing related overheads and complexity compared to traditional global checkpointing methods, but faces coordination complexity and challenges in handling partial failures. Simon *et al.* [21] introduces delayed CIC (DCIC) based FTM to improve performance in distributed systems by strategically placing checkpoints based on communication delays rather than fixed intervals. This approach enhances fault tolerance and self-healing capabilities and minimizes the performance overheads but encounters

useless checkpoints. This approach is also not addressing overall fault scenarios in distributed systems. Roberto Baldoni *et al.* [5] propose an index-based CIC algorithm to reduce checkpointing overheads, enhance efficiency in distributed systems without centralized control. Despite its innovative approach, the algorithm faces limitations such as potential overhead, implementation complexities, scalability concerns, limited fault model coverage, significant recovery time and resource utilization issues.

Luo and Manivannan [27] introduces a fully informed and efficient (FINE) CIC protocol designed to minimize checkpointing overhead by reducing forced checkpoints and avoiding the domino effect, ensuring coordinated recovery through a fully informed global state. This approach still faces implementation complexity issues along with communication overhead and network dependency, in addition. Ahn [9] proposes a FTM that combines CIC with message logging to overcome the limitations in existing FTMs in distributed systems. His methodology involves forcing checkpoints based on communication patterns and logging all messages for replay during recovery and advanced mechanisms to handle dependencies. Despite its benefits, the approach faces useless checkpoint overheads, recovery latency and dependency management issues. Malhotra and Bala [17] propose a CIC FTM tailored for IoT systems. The protocol is a combination of spontaneous checkpointing where each node takes spontaneous checkpoints based on a logistic function that estimates the time interval between checkpoints and coordinated checkpointing where nodes take coordinated checkpoints using the Takagi-Sugeno (T-S) fuzzy system, which generates results based on definite-39 rules. This system uses parameters such as energy, failure rate and received signal strength indicator (RSSI) to avoid unnecessary checkpoints. This approach minimizes the number of checkpoints, system overhead and ensuring non-blocking processes during checkpointing. In this permanent checkpoints are stored on IoT devices which also indirectly leads to considerable useless checkpoints after fault recovery and also fault recovery based on specific parameters dependency and since, this protocol is designed specifically for mobile distributed systems, has its performance is highly dynamic or large-scale networks with significant number of nodes is not thoroughly evaluated. Jaggi and Singh [13] introduce an adaptive checkpointing technique to enhance fault tolerance in mobile computing grid (MoG). This approach uses cooperative checkpointing, where nodes in the system cooperate to store checkpoint data. If a node lacks stable storage, it uses the storage of other nodes. The checkpoint data is replicated across multiple nodes to ensure higher chances of recovery depending on resource availability. The checkpointing scheme adapts based on the availability of resources in the MoG making it flexible and efficient. But, this approach highly relies on the availability of stable storage and resources in other nodes, which might not always be guaranteed in highly dynamic or resource-constrained environments. Replicating checkpointing data at multiple nodes can lead to increased network traffic and overhead and as the number of nodes increases, managing and coordinating checkpoints across a large number of nodes may become complex and less efficient.

Tan *et al.* [32] uses long short-term memory (LSTM) networks for detecting faults in non-linear dynamical systems. The methodology involves training LSTM model on normal operating data, using it to predict future states and detecting predictive faults. This approach can be redesigned for different systems on requirement basis even though it is challenging. Absar *et al.* [33] assess the use of LSTM models for forecasting contagious disease outbreaks, focusing on utilizing historical data for early detection. The methodology involves training LSTM models on disease data, making predictions and evaluating the model's performance using accuracy metrics. Zhang *et al.* [34] introduce an LSTM-based autoencoder for network anomaly detection aiming to identify unusual traffic patterns with high reconstruction errors and reconstruct normal network traffic. This approach not feasible with respect to practical applicability and integration in real-world scenarios. Wang *et al.* [35] focus to improve prediction accuracy and efficiency of an LSTM model used for predicting temperature in data centers by optimizing its hyperparameters. This hyperparameters optimization technique can be utilized based on domain and requirement specific since it adds on computational costs.

The detailed survey outlines that coordinated and un-coordinated checkpointing encounters checkpointing overheads and inefficiencies. While addressing the challenges in optimizing the checkpointing based FTMs, the existing techniques like DCIC encounters useless checkpoints, index-based CIC algorithms face issues like checkpointing overheads and implementation complexities. Adaptive checkpointing in mobile computing grids is resource-dependent and may not always be feasible in dynamic environments like IoT. Protocols tailored for IoT systems need to minimize number of checkpoints, useless checkpoints and efficiently recover transient faults with minimal resource utilization. The main contributions of this research are as follows:

- The research introduces a novel checkpoint at intermediate node (CIN) CIC-FTM for IoT environments that strategically place checkpoints only just before predicted faulty nodes by integrating LSTM-deep learning algorithms. This minimizes useless and forced checkpoints, unlike traditional methods that place checkpoints indiscriminately across all/many nodes, and also addresses the deficiencies of existing FTMs.
- Describing transient fault occurrence scenarios and implementation of the fault detection algorithms.

- Our approach uses LSTM-deep learning for training using history of fault occurrences and new message logs and predicting new fault occurrence.

The remaining sections are arranged as follows: section 2 describes the proposed methodology that includes transient fault detection, LSTM algorithm for predicting fault occurrence and CIN CIC-FTM operational mechanism for placing checkpoints and fault recovery. Section 3 explains experimental setup, section 4 about results analysis and finally section 5 concludes the research work.

2. METHOD

The proposed methodology integrates novel CIN CIC-FTM that strategically place checkpoints based on LSTM predictions that minimizes checkpointing overheads and enhances system performance by transient fault recovery. The step-by-step description starts with system architecture diagram, Figure 2 starts from data collection, pre-processing that include handling missing values, removing outliers and ensuring consistency, transient fault detection, prediction of fault occurrences using LSTM and checkpoint placement and fault recovery using CIN CIC-FTM.

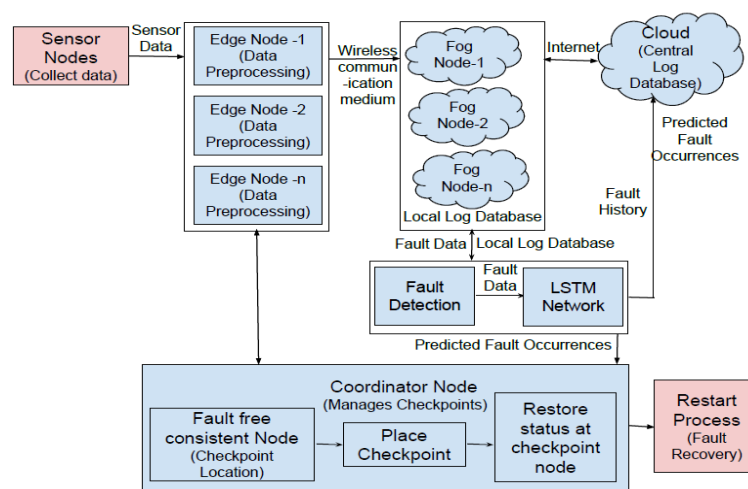


Figure 2. Block diagram of CIN CIC-FTM using LSTM

Data collection and processing: initially, the sensors readings like temperature, humidity, water level, light and moisture in IoT agriculture scenario, are configured and collected every minute (adaptable based on application requirement). We simulate the proposed model on Google Colab using Python 3.7, utilizing Python libraries including Pandas, OS, Pickle, Socket, Numpy, Crcmode, Tracemalloc, Psutil, Matplotlib, Keras and TensorFlow. Messages between processes were generated using exponential distribution spanning between 4 to 100 processes. Periodically save the state of each sensor/edge node and communication state using ‘Pickle’ library to serialize and store the state. Messages exchanged using ‘Socket’ library between nodes were logged in local and central database to enable recovery. A coordinator node is designated to manage checkpointing.

Transient fault detection: fault detection algorithms (refer Algorithm 1 to 4) are implemented to identify transient faults. Transient faults are software faults and temporary that occur during runtime of the system. Transient faults appear to be unpredictable for a very short period of time whose impact is anomalously large on IoT system while decision-making and actuating. Also, frequency of these transient fault occurrences is exponentially proportional to abnormality in system actions [4], [8]-[10], [12], [36]. In IoT applications the most frequently appearing transient faults are i) if there is incorrect/missing sequence number in packet of information flow, the system responds differently, and then the sequence number fault – F1 is detected. The information is of two types – payload and command sequence number. ii) If integrity of the information is affected, checksum fault – F2 is detected and checksum based fault detection technique is used. iii) Integrity of the information is also verified by checking the length of the message. If the length of the message received or transmitted is not as expected or received data is irrelevant with its previously generated pattern, then such fault is represented as null character fault – F3. (iv) When the information received or transmitted is beyond the payload range, the out of range fault – F4 is detected. The transient faults detected using the fault detection algorithms affect the fault occurrence scenarios by designing occurrence semantics in my work. The detected faults, their occurrence and fault states are logged.

Algorithm 1. To detect fault F1

```

Initialize: expected_seq_num = 0
tolerance_threshold = 5 // Adjust as needed
Function
detect_seq_num_fault(received_seq_num):
    if received_seq_num == expected_seq_num:
        // Received sequence number is as
        expected
        expected_seq_num += 1
        return NO_FAULT
    else if received_seq_num < expected_seq_num:
        // Received a duplicate sequence number
        return "Duplicate Sequence Number Detected"

```

Algorithm 2. To detect fault F2

```

Function calculate_chksum(data):
//Calculate a checksum value for the given
data
    chksum = 0
    for each byte in data:
        chksum += byte
    // Add the value of each byte to the
    checksum
    return checksum
Function
detect_chksum_fault(received_data,
received_chksum):
    expected_chksum =
calculate_chksum(received_data)
    if received_chksum == expected_chksum:
        //Checksum matches, data is intact
        return NO_FAULT
    else:
        //Checksum mismatch, data is corrupted
        return "Checksum Mismatch Detected"
    fault =
detect_chksum_fault(received_data,
received_chksum)
if fault != NO_FAULT: print(fault)

```

Algorithm 4. To detect fault F4

```

//For different types of sensors it is
essential to configure range values. Based
on configuration, error is detected
// Initialize variables
expectedMinValue = 0, expectedMaxValue =
100
receivedMinValue = 0, receivedMaxValue =
0
// Function to check if the received data is
within the expected range
function isDataInRange(data):
    if data >= expectedMinValue and data
<= expectedMaxValue:
        return true
    else:
        return false
// Function to calculate the checksum
function calculateChecksum(data):
    checksum = 0
    for value in data:
        checksum += value
    return checksum % 256
// Main program loop
while true:
    // Wait for incoming message
    message = receiveMessage()
    // Extract the data, received minimum value,
received maximum value, and checksum from
the message
    data = extractData(message)
    receivedMinValue =
extractMinValue(message)

```

```

    else if received_seq_num -
expected_seq_num > tolerance_threshold:
        // Sequence number gap exceeds the
        tolerance threshold
        return "Sequence Number Gap Detected"
    else:
        // Sequence number gap is within the
        tolerance threshold
        expected_seq_num =
received_seq_num + 1
        return NO_FAULT

```

Algorithm 3. To detect fault F3

```

while true:
    // Wait for incoming data
    trg_data = receiveData()
    src_data = sendData()
    // Check for null character
    if hasNullCharacter(src_data, trg_data):
        logError("Null character detected in the
transmitted data")
    else:
        // Null character not detected, process the
        data
        processData(data)
    // Function to check for null character
function
hasNullCharacter(src_data, trg_data):
    if length(src_data) !=
length(trg_data):
        return true
    for character in trg_data:
        if character == '\0'
        return true
    return false

```

```

receivedMaxValue =
extractMaxValue(message)
receivedChecksum =
extractChecksum(message)
// Verify the checksum
calculatedChecksum =
calculateChecksum(data)
if calculatedChecksum ==
receivedChecksum:
    // Checksum is valid, now check for data
range
    if receivedMinValue >=
expectedMinValue and receivedMaxValue <=
expectedMaxValue:
        // Data range is within the expected range
        if isDataInRange(data):
            // Data is within the expected range,
            process the data
            processData(data)
        else:
            // Data is out of range
            logError("Received data
is out of range")
    else:
        // Received range is out of the expected
range
        logError("Received data range
is out of the expected range")
    else:
        // Checksum is invalid
        logError("Invalid checksum
detected")

```

Prediction of fault occurrence: the LSTM network takes collected sequential input data at various time stamps and historical data of fault occurrences for training purposes and trained with multiple epochs to predict the next faulty node. This predicted output is logged as a historical dataset on every new prediction. Dataset also includes timestamps and labels indicating whether a fault occurred. Split the dataset into training, validation and test sets. The training set is used to train the LSTM model, the validation set is used to tune hyperparameters and the test set is used to evaluate the model. The LSTM model has an input layer, layers of LSTM and a dense output layer. Train the LSTM model on the training data and validate it using the validation data. The model is evaluated on the test data using relevant metrics as accuracy, precision, recall and F1-score.

LSTM, an advanced recurrent neural network (RNN) version, retains vital data, discarding less useful. LSTM includes input, forget and output gates [34]. The forget gate, represented by “ f_t ”, discards irrelevant data. It’s determined by a sigmoid function based on weights and bias. Input “ X_t ” feeds into the network, with the forget gate deciding if “ h_{t-1} ” output is relevant to current data. Activation function “ $(h_{t-1}+X_t)$ ” determines relevance, closer to zero implies irrelevance, otherwise relevance. Figure 3 shows the LSTM unit with its components.

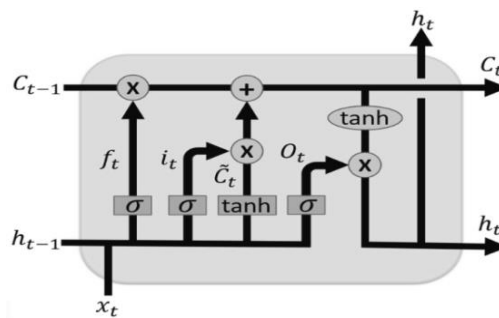


Figure 3. Basic LSTM cell

The “ f_t ” is a forget gate function, that forgets, or ignores the irrelevant information. “ W_{t_f} ”, are the weights and “ bs_{t_f} ” the bias of the forget gate, σ is a sigmoid curve, and “ \cdot ” implies multiplication of the matrix. The next step is to determine, using two procedures, which new knowledge is present in the cell state. The input gate first determines which states are updated, and then the “ \tanh ” activation function is used to generate a vector of potential new values. The current cell state “ C_t ” is then changed using outputs from the input gate, the forget gate, and the “ \tanh ” layer. Finally, the output gate and a “ \tanh ” function are used to calculate the output of network “ h_t ”, Where “ W_{t_o} ”, is the input weight and “ bs_{t_o} ” the bias of the output gate, respectively.

$$f_t = \sigma(W_{t_f} \cdot [h_{t-1}, X_t] + bs_{t_f}) \quad (1)$$

$$i_t = \sigma(W_{t_i} \cdot [h_{t-1}, X_t] + bs_{t_i}) \quad (2)$$

$$\sim C_t = \tanh(W_{t_c} \cdot [h_{t-1}, X_t] + bs_{t_c}) \quad (3)$$

$$C_t = f_t * C_{t-1} + i_t * \sim C_t \quad (4)$$

$$O_t = \sigma(W_{t_o} \cdot [h_{t-1}, x_t] + bs_{t_o}) \quad (5)$$

$$h_t = O_t * \tanh(C_t) \quad (6)$$

$$y_{t+n} = o_n(y_t, y_{t+1}, y_{t+2}, \dots, y_{t+n-1}) \quad (7)$$

Dropout regularization combats over-fitting by removing certain neurons. Each hidden unit in a neural network is trained using a random sample of others. LSTM networks predict ‘n’ time steps ahead, contrasting projected with actual values. Input data “ X_t ” at time stamp “t” is processed sequentially with a sliding window method. LSTM predicts the next faulty node within the e (size-15). The first sample of input is x_1 , second is x_2 and so $X_t = \{x_{t-1}, x_t, x_{t+1}\}$. This input taken at “ W_L ” window length is $\{x_{t-w}, x_{t+1-w}, \dots, x_{t-1}\}$ used for prediction. The LSTM network is trained with “n” epochs and “ Y_t ” is output – predicted faulty node and “ O_n ” is LSTM network

output. LSTM's advantage lies in predicting future values while remembering past ones, until the required multistep predictions are made.

Checkpoint placement and fault recovery using CIN CIC-FTM: the core of the CIN CIC-FTM is the prediction of potential fault occurrences using LSTM network. The predicted faults occurrences are logged as fault history. Initially, local checkpoints are placed at nodes after significant system events. Later after training LSTM model, forced checkpoints are placed at latest consistent node prior to predicted faulty node. This is done dynamically, with the model continuously updating its predictions as new data is processed. The checkpointing process involves saving the current state of the system, including the process data and current status of the messages being processed. The system upon fault detection checks whether the current node or its preceding node has valid checkpoints. The system initiates a rollback to the most recent checkpoint, allowing the process to restart from the known good (fault free) state and preventing the spread of faults through the system. After successful recovery and the stabilization of the process, the forced checkpoints that are already logged as fault history are erased from the local memory and the system continues to operate with the LSTM model adjusting its predictions based on the fault history. This approach specifically reduce the storage overheads because of checkpoints. The algorithm is given in Algorithm 5 and working principle is explained as shown in Figure 4.

Algorithm 5. CIN CIC-FTM

```

import libraries: numpy, tensorflow,
sequential, LSTM, dense, pickle, os
# Load historical data for fault
prediction
def load_dta(file_path):
    load historical data related to
    transient faults
    data = np.load(file_path)
    return data
# Build and train LSTM model
def build_lstm_model(input_set): model =
Sequential()
# predict fault occurrences
def predict_faults(model, x_input):
    predictions = model.predict(x_input)
    return predictions
# Place checkpoints based on predictions
def place_checkpoints(nodes,predictions,
threshold=0.65):
    checkpoints = [ ]
    for i, predictions in
enumerate(predictions):
        if predictions > threshold:
            checkpoints.append(nodes[i])
    return checkpoints
# Save system state as checkpoint
def save_checkpoint(node, state,
checkpoint_dir):
checkpoint_path=os.path.join(checkpoint_dir, f'checkpoint_node_{node}.pkl')
    with open(checkpoint_path, 'wb') as
file:
        pickle.dump(state, file)
# Fault detection
def
detect_faults(nodes,predictions,current_state):
    faulty_nodes = [ ]
    for i, node in enumerate(nodes):
        if predictions[i] > 0.65 and
current_state[node]== 'faulty':
            faulty_node.append(node)
    return faulty_nodes
# Fault recovery using checkpoints
def recover_from_fault(node,
checkpoint_dir):
checkpoint_path=os.path.join(checkpoint_dir, f'checkpoint_node_{node}.pkl')
    with open(checkpoint_path, 'rb') as file:
        state = pickle.load(file)
    return state
model.add(LSTM(50,activation='relu',
input_set=input_set))
model.add(Dense(1))
model.compile(optimizer = 'adam', loss =
'mse')
    return model
def train_lstm_model(model,x_train,y_train,
epochs=1000):
    model.fit(x_train, y_train, epochs =
epochs, erbose =1)
    return model
# Main process to run CIN CIC-FTM
def
cin_cic_ftm_process(nodes,fault_data_file,
checkpoint_dir, current_state):
    data = load_data(fault_data_file)
    x_train, y_train = data['x_train'],
data['y_train']
    input_set = (x_train.set[1],
x_train.set[2])
    # Build and train the LSTM model
    model = build_lstm_model(input_set)
    model = train_lstm_model(model,
x_train, y_train)
    # Predict faults
    predictions = predict_faults(model,
x_train)
    # Place checkpoints based on predictions
    checkpoints= place_checkpoints(node,
predictions)
    for node in checkpoints:
        state = current_state[node]
        save_checkpoint(node, state,
checkpoint_dir)
    # Detect faults
    faulty_nodes=detect_faults(nodes,predictions,
current_state)
    # Recover from faults
    for node in faulty_nodes:
        recovered_state=recover_from_fault(node,
checkpoint_dir)
        current_state[node] = recovered_state
    return current_state
# Example usage
nodes = ['node1', 'node2', 'node3',
'node4']
fault_data_file = 'fault_data.npz'
checkpoint_dir = 'checkpoints'
current_state = {'node1': 'normal',
'node2': 'normal', 'node3': 'normal',
'node4': 'normal'}
final_state=cin_cic_ftm_process(nodes,
fault_data_file, checkpoint_dir,
current_state)

```

This algorithm provides a detailed framework for implementing the CIN CIC-FTM in Python, enabling efficient fault prediction, strategic checkpoint placement and fault recovery in IoT applications. The 'load_data' function loads historical fault data for training the LSTM model. The data is expected to be in a '.npz' file containing training and testing datasets. The LSTM model is built using TensorFlow's Keras API, with an input layer, LSTM layer and output dense layer. The model is trained on the historical fault data. The checkpoints are placed on nodes where the predicted fault probability exceeds a certain threshold (0.65). The state of these nodes is saved using the 'pickle' library. The main function 'cin_cic_ftm_process' integrates all steps and processes the nodes in the IoT network to place checkpoints and perform fault recovery.

Consider Figure 4 as an example for understanding CIN CIC-FTM. Firstly, CIN based CIC induces checkpoints C1 and C2 at node N_{2,1} and N_{3,2} as these nodes are predicted to have faults based on LSTM algorithm. The sequence of message transmission is- message M1 is sent from process P4 to P3, message M2 from P1 to P3, and so on. If at M5, a fault is detected, CIC-FTM initiates the rollback propagation [37] and will rollback to the latest placed checkpoint, C1 and restarts from C1. Again, if fault is not recovered at M5, the system will rollback to next latest checkpoint C2 and restarts from there. Still if fault is not recovered, then FTM returns to initial states and restarts the process from initial states. Still if fault is not recovered, the FTM concludes its hardware fault.

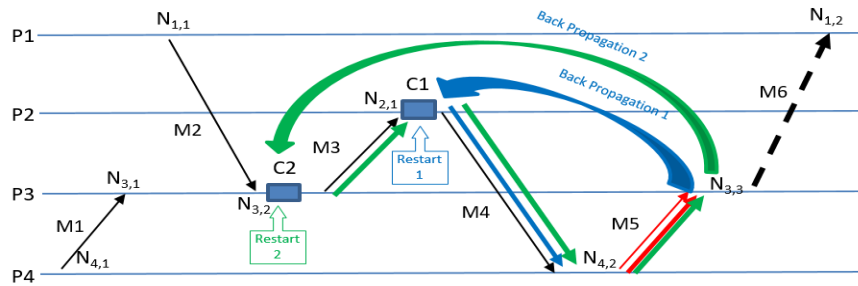


Figure 4. Example for checkpoint at intermediate node based CIC-FTM

3. RESULT AND DISCUSSION

In this section, we simulate the proposed model on the Google Colab platform using Python 3.7. Discrete event simulation spans 4-100 processes, generating messages via exponential distribution. Sensor frequency, set to one minute, adaptable based on application requirements. The sensors used in IoT agriculture include temperature, humidity, water level, light, and moisture sensors. Training involves 7000+ dataset, 1000+ for testing under diverse scenarios. Each scenario undergoes 1-15 tests, averaging parameters for assessment. Reconfiguring the sensor frequency to suit the needs of the application is possible.

The performance parameters for prediction of fault nodes using LSTM model are i) precision, "P", is the percentage of values identified precisely; ii) recall, "R", which is the number of positive class prediction out of all positive examples; and iii) F1-score, "F1", which is the harmonic mean of precision and recall, provides a more accurate picture of cases that were incorrectly classified than the accuracy metric. The accuracy is defined as in expression (11), where true negative="TN", true positive="TP" and false positive="FP", false negative="FN". Among the various loss functions, root mean squared error (RMSE) is the appropriate metric to evaluate time series [38] prediction models and it is defined as in expression (12), where, "n" is number of nodes, "i" is number of iterations, "y" is real output, "yⁱ" is desired output. The larger RMSE indicates less accuracy and least RMSE indicates highest accuracy [39]:

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{8}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{9}$$

$$F1\ Score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{10}$$

$$(Accuracy, Error\ Rate) = \left(\left(\frac{TP + TN}{TP + TN + FP + FN} \right), \left(\frac{FN + FP}{TP + FP + TN + FN} \right) \right) \tag{11}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y^i - y_p^i)^2} \tag{12}$$

Hyperparameters are used to tune the LSTM based prediction models. The most important hyperparameter combination that have direct impact on model performance are as mentioned in Table 1 with the threshold value of 0.65 for which the prediction accuracy obtained is 98.5% as tabulated in Table 2 and shown in Figure 5. The integration of the LSTM-based deep learning algorithm showed high accuracy in predicting fault occurrences as shown in Figure 5. The LSTM model was trained and validated using historical fault data and it successfully identified transient fault occurrences with an accuracy of 98.5% accuracy, as measured by standard metrics including precision, recall and F1-score as tabulated in Table 2.

Table 1. Hyperparameter values for highest accuracy

Hyperparameters	Values
Window-size	15
Dropout-rate	0.2
Regularizer	L2
Regularizer-rate	0.014
Epochs	1000
Activation-function	ReLu
Learning-rate	0.012

Table 2. Average performance metrics of LSTM model

Epochs	% Accuracy	Recall	F1-score
50	98.4	98.2	98
100	98.5	98.23	98.12
200	98.6	98.4	98.2
500	98.5	98.2	98.1
1000	98.6	98.3	98
Average (from 50 to 1000 epochs)	98.52	98.23	98.1

An average recall of 98.23% indicated that, out of all the actual fault occurrences, the LSTM model successfully predicted 98.23% of them and this high recall value indicates that the model is very effective at detecting most of the faults that occur, minimizing the risk of unpredicted faults which leads to system failures. An F1-score of 98.1% indicates that the LSTM model has a well-balanced performance, with both high recall and high precision. This indicates the model is reliable in predicting faults with minimal false alarms, which is critical for efficient fault management in IoT systems. Now, the proposed CIN CIC-FTM places the checkpoints at the identified faulty nodes. The Figure 6 shows average number of checkpoints placed, average number of useless checkpoints, memory consumption and CPU consumption for checkpointing in the network, and average back propagation time and fault recovery time, when the network is of different node sizes like, 4 nodes, 10 nodes, 40 nodes and 100 nodes.

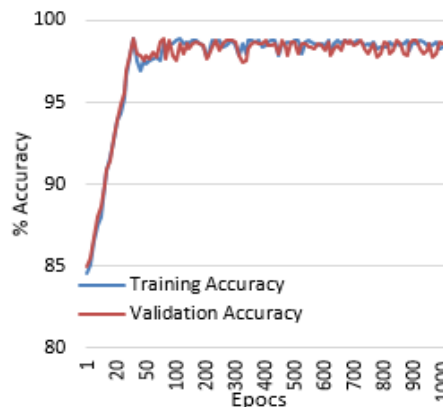


Figure 5. LSTM fault prediction accuracy

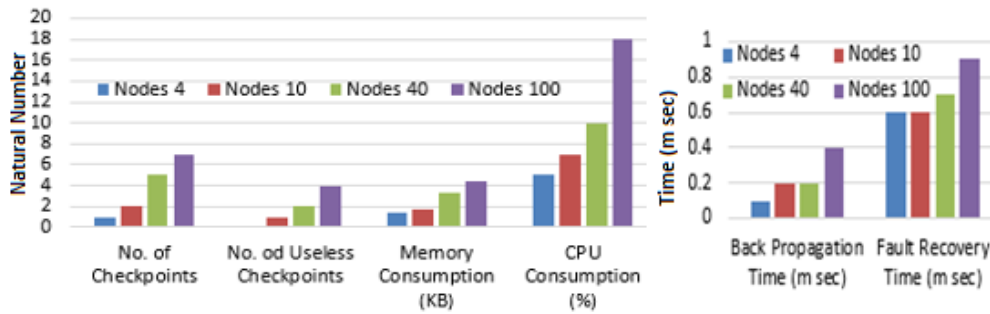


Figure 6. Checkpoints count and resource consumption by CIN CIC-FTM

The observations are:

- The CIN CIC-FTM using LSTM places checkpoints at predicted faulty nodes that are 15% of the total number of nodes in the network, hence significantly reduces the number of forced and useless checkpoints compared to existing methods [9], [17], [21], [22], [39] that often place checkpoints indiscriminately across multiple nodes.
- The CIN CIC-FTM makes targeted checkpoint placement just before nodes predicted to be faulty. These checkpoints become useless only if fault recovery is successful by restarting from the most recently placed checkpoint, preventing the need to rollback to subsequent ones. As a result, the number of useless checkpoints is minimized-less than 10% of the total number of checkpoints.
- Existing checkpointing methods involve frequent, periodic and indiscriminate checkpoint placements [5], [9], [13], [17], [21], [22], [24], [25], [27], [32], [39], leading to unnecessary memory consumption due to the storage of multiple and useless checkpoints. The CPU is loaded with the continuous monitoring and processing of these checkpoints, reducing system efficiency [5], [9], [13], [17], [39]. In contrast, the CIN CIC-FTM, since the checkpoints are placed at targeted nodes based on high accurately predicted nodes, minimizes the memory consumption- by reducing the number of checkpoints stored and the CPU consumption- as the system is not required to constantly process and manage an excessive number of checkpoints.
- IoT networks face challenges with respect to its dynamic behavior and resource constraints [5], [9], [13], [22], [25], [27], [39]. The reduction in memory and CPU consumption not only impress the overall efficiency of the system but also ensures that the IoT applications can continue to operate effectively even in resource-limited scenarios.
- Back propagation time refers to the time required to rollback to the recently placed checkpoint from the fault detected node. Fault recovery time refers to the time taken to recover from a fault, by loading the previous consistent checkpoint and resuming execution from that checkpoint onwards. It includes the time required to transfer data from checkpoint storage to the program's memory, reinitialize the execution state and continue its execution. The results indicate that the CIN CIC-FTM reduces the back propagation time and fault recovery time by considerable margin compared to existing checkpointing techniques, which suffer from inefficiencies due to inconsistent node states and the domino effect [5], [9], [13], [24], [39], [40]. Because of predicted and targeted checkpointing approach, the overall number of checkpoints are reduced and are placed just before nodes predicted to be faulty, the back propagation time and fault recovery time are significantly reduced in turn allowing for quicker system response, even in presence of transient faults.

4. CONCLUSION

The increasing reliance on IoT applications across various sectors such as healthcare, industry and agriculture focus on the critical need for robust FTMs. Transient software faults pose significant risks to the performance and reliability of these systems. The existing FTMs struggle with inefficiencies related to checkpointing overheads, as checkpoints are typically placed frequently, periodically and indiscriminately across all nodes, leading to useless and forced checkpoints, followed by domino effect and unnecessary resource consumption. The research introduces a novel CIN based CIC-FTM that incorporates LSTM deep learning algorithms to predict fault occurrences and strategically place checkpoints before the predicted faulty nodes. The prediction accuracy obtained is 98.5%. This reduces the number of checkpoints placed, that is at <15% nodes of the overall nodes and <10% of them go useless, which proportionally reduces rollback time

and fault recovery time as discussed in results section. Hence the storage overheads, rollback-restart-recovery process overheads and overall checkpointing overheads, are significantly reduced. The CIN CIC-FTM also ensures that IoT systems can operate more efficiently, even in resource-constrained scenarios, making the CIC-FTM a promising solution for enhancing the robustness of IoT applications against transient faults. The proposed FTM is flexible to i) rescale for more than 100 node and complex IoT network with a high density interconnected devices and ii) other IoT applications like smart cities, healthcare and industrial automation. To address the dynamic nature of IoT environments, future research could focus on developing adaptive and self-learning FTMs. By continuously learning from new data and adapting the checkpointing strategy considering this research as the baseline approach, the system could improve its fault prediction accuracy and efficiency over time.

ACKNOWLEDGEMENT

The authors would like to take this opportunity to acknowledge, the Doctoral Committee members for their guidance and Sir. M. Visvesvaraya Institute of Technology, Bengaluru, India for providing the necessary facilities to support this research work.




REFERENCES

- [1] K. Ashton, "That internet of things thing," *RFID Journal*, vol. 22, no. 7, pp. 97–114, 2009.
- [2] F. Cristian, "Understanding fault-tolerant distributed systems," *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, Feb. 1991, doi: 10.1145/102792.102801.
- [3] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "BlueGene/L failure analysis and prediction models," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2006, pp. 425–434, doi: 10.1109/DSN.2006.18.
- [4] S. Kalaiselvi and V. Rajaraman, "A survey of checkpointing algorithms for parallel and distributed computers," *Sadhana*, vol. 25, no. 489–510, 2000.
- [5] R. Baldoni, F. Quaglia, and P. Fornara, "An index-based checkpointing algorithm for autonomous distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 2, pp. 181–192, 1999, doi: 10.1109/71.752783.
- [6] I. P. Egwuotuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *Journal of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013, doi: 10.1007/s11227-013-0884-0.
- [7] E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002, doi: 10.1145/568522.568525.
- [8] R. Garg and P. Kumar, "A review of fault tolerant checkpointing protocols for mobile computing systems," *International Journal of Computer Applications*, vol. 3, no. 2, pp. 8–19, 2010, doi: 10.5120/710-998.
- [9] J. Ahn, "Communication-induced checkpointing with message logging beyond the piecewise deterministic (PWD) model for distributed systems," *Electronics*, vol. 10, no. 12, p. 1428, Jun. 2021, doi: 10.3390/electronics10121428.
- [10] B. H. Sababha and O. A. Rawashdeh, "Evaluation of communication induced checkpointing in resource constrained embedded systems," in *Proceedings of the ASME Design Engineering Technical Conference*, 2011, vol. 3, no. PARTS A AND B, pp. 39–45, doi: 10.1115/DETC2011-48634.
- [11] H.-D. Ma, "Internet of things: objectives and scientific challenges," *Journal of Computer Science and Technology*, vol. 26, no. 6, pp. 919–924, Nov. 2011, doi: 10.1007/s11390-011-1189-5.
- [12] J. Tsai, S.-Y. Kuo, and Y.-M. Wang, "Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 963–971, 1998.
- [13] P. K. Jaggi and A. K. Singh, "Adaptive checkpointing for fault tolerance in an autonomous mobile computing grid," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, Nov. 2014, pp. 553–557, doi: 10.1109/IC3I.2014.7019711.
- [14] G. H. Adday, S. K. Subramaniam, Z. A. Zukarnain, and N. Samian, "Fault tolerance structures in wireless sensor networks (WSNs): survey, classification, and future directions," *Sensors*, vol. 22, no. 16, p. 6041, Aug. 2022, doi: 10.3390/s22166041.
- [15] A. Sowjanya Lakshmi, C. Vani Priya, and G. Gupta, "Communication induced checkpointing based fault tolerance mechanism – a review and CIAC-FTM framework in IoT environment," in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Nov. 2022, pp. 1–6, doi: 10.1109/ICCCIS56430.2022.10037637.
- [16] A. Sari and M. Akkaya, "Fault tolerance mechanisms in distributed systems," *International Journal of Communications, Network and System Sciences*, vol. 08, no. 12, pp. 471–482, 2015, doi: 10.4236/ijcns.2015.812042.
- [17] N. Malhotra and M. Bala, "Fault-tolerant communication induced checkpointing and recovery protocol using IoT," *Tech Science Press, Intelligent Automation & Soft Computing*, vol. 30, no. 3, pp. 945–960, 2021, doi: 10.32604/iasc.2021.019082.
- [18] M. T. Moghaddam and H. Muccini, "Fault-tolerant IoT: a systematic mapping study," in *Software Engineering for Resilient Systems: 11th International Workshop*, 2019, pp. 67–84, doi: 10.1007/978-3-030-30856-8_5.
- [19] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. De Mel, "An analysis of communication induced checkpointing," in *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 99CB36352)*, 1999, pp. 242–249, doi: 10.1109/FTCS.1999.781058.
- [20] M. Castro-León, H. Meyer, D. Rexachs, and E. Luque, "Fault tolerance at system level based on RADIC architecture," *Journal of Parallel and Distributed Computing*, vol. 86, pp. 98–111, 2015, doi: 10.1016/j.jpdc.2015.08.005.
- [21] A. C. Simón, S. E. Pomares Hernandez, J. R. Perez Cruz, R. Ben Halima, and H. Hadj Kacem, "Self-healing in autonomic distributed systems based on delayed communication-induced checkpointing," *International Journal of Autonomous and Adaptive Communications Systems*, vol. 9, no. 3–4, pp. 183–200, 2016, doi: 10.1504/IJAACS.2016.079621.
- [22] I. C. Garcia and L. E. Buzato, "Checkpointing using local knowledge about recovery lines." University of Campinas, Brazil, pp. IC-99–22, 1999.
- [23] J. Dongarra, T. Herault, and Y. Robert, "Fault tolerance techniques for high-performance computing," in *Computer Communications and Networks*, Springer, Cham, 2015, pp. 3–85.




- [24] P. Eles, V. Izosimov, P. Pop, and Z. Peng, "Synthesis of fault-tolerant embedded systems," in *2008 Design, Automation and Test in Europe*, Mar. 2008, pp. 1117–1122, doi: 10.1109/DATE.2008.4484825.
- [25] J.-M. Hélary, A. Mostefaoui, R. H. B. Netzer, and M. Raynal, "Communication-based prevention of useless checkpoints in distributed computations," *Distributed Computing*, vol. 13, no. 1, pp. 29–43, Jan. 2000, doi: 10.1007/s004460050003.
- [26] J.-M. Helary, A. Mostefaoui, and M. Raynal, "Communication-induced determination of consistent snapshots," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 865–887, 1999, doi: 10.1109/ftcs.1998.689472.
- [27] Y. Luo and D. Manivannan, "FINE: a fully informed and efficient communication-induced checkpointing protocol for distributed systems," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 153–167, 2009, doi: 10.1016/j.jpdc.2008.07.012.
- [28] H. Y. Teh, A. W. Kempa-Liehr, and K. I.-K. Wang, "Sensor data quality: a systematic review," *Journal of Big Data*, vol. 7, no. 1, p. 11, Dec. 2020, doi: 10.1186/s40537-020-0285-1.
- [29] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proceedings of the 12th USENIX Security Symposium*, 2003, pp. 169–186.
- [30] I. C. Garcia, G. M. D. Vieira, and L. E. Buzato, "A rollback in the history of communication-induced checkpointing," *arXiv preprint arXiv: 1702.06167*, 2019, [Online]. Available: <http://arxiv.org/abs/1702.06167>.
- [31] G. M. D. Vieira, G. M. D. Vieira, I. C. Garcia, L. E. Buzato, and L. E. Buzato, "Systematic analysis of index-based checkpointing algorithms using simulation," in *SCTF '01: Proc. of the IX Brazilian Symposium on Fault-Tolerant Computing*, 2001, pp. 31–42.
- [32] Y. Tan, C. Hu, K. Zhang, K. Zheng, E. A. Davis, and J. S. Park, "LSTM-based anomaly detection for non-linear dynamical system," *IEEE Access*, vol. 8, pp. 103301–103308, 2020, doi: 10.1109/ACCESS.2020.2999065.
- [33] N. Absar, N. Uddin, M. U. Khandaker, and H. Ullah, "The efficacy of deep learning based LSTM model in forecasting the outbreak of contagious diseases," *Infectious Disease Modelling*, vol. 7, no. 1, pp. 170–183, Mar. 2022, doi: 10.1016/j.idm.2021.12.005.
- [34] L. Zhang, W. Da Zhou, P. C. Chang, J. W. Yang, and F. Z. Li, "Iterated time series prediction with multiple support vector regression models," *Neurocomputing*, vol. 99, pp. 411–422, 2013, doi: 10.1016/j.neucom.2012.06.030.
- [35] S. Wang, C. Ma, Y. Xu, J. Wang, and W. Wu, "A hyperparameter optimization algorithm for the LSTM temperature prediction model in data center," *Scientific Programming*, vol. 2022, pp. 1–13, Dec. 2022, doi: 10.1155/2022/6519909.
- [36] F. Quaglia, R. Baldoni, and B. Ciciani, "On the no-Z-cycle property in distributed executions," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 400–427, 2000, doi: 10.1006/jcss.2000.1720.
- [37] J. Brzeziński, J.-M. Helary, and M. Raynal, "Semantics of recovery lines for backward recovery in distributed systems," *Annales Des Télécommunications*, vol. 50, no. 11–12, pp. 874–887, 1995, doi: 10.1007/bf03005244.
- [38] M. Said Elsayed, N. A. Le-Khac, S. Dev, and A. D. Jurcut, "Network anomaly detection using LSTM based autoencoder," in *Q2SWinet 2020 - Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, 2020, pp. 37–45, doi: 10.1145/3416013.3426457.
- [39] B. K. Saraswat, R. Suryavanshi, and D. Yadav, "A comparative study of checkpointing algorithms for distributed systems," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 20, pp. 1595–1603, 2018.
- [40] D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino effect free recovery algorithm," *Symposium on Reliability in Distributed Software and Database Systems*, vol. 84, pp. 207–215, 1984.

BIOGRAPHIES OF AUTHORS



Sowjanya Lakshmi A    is a Research Scholar pursuing Ph.D. at Research Center of Department of Computer Science and Engineering, Sir. M. Visvesvaraya Institute of Technology, Bengaluru, India under Visvesvaraya Technological University, Belagavi, India. Her specialization area is internet of things, artificial intelligence, machine learning, and data analytics. She has more than 13 years of experience in teaching and research and currently working as Assistant Professor in the Department of Information Science and Engineering. She has published papers in National and International Journals. She can be contacted at email: sowjanya.egg@gmail.com.



Vanipriya Ch    is Professor and Head of the department of MCA, Sir. M. Visvesvaraya Institute of Technology, Bengaluru, India. She holds a Ph.D. degree in Computer Science and Engineering with specialization in Sentimental Analysis. She has 23 years of Teaching and Research. Her research areas are artificial intelligence, machine learning, data mining, and sentimental analysis. She has patent on her innovative idea and published papers in Scopus indexed Journals. She can be contacted at email: vanipriya_is@sirmvit.edu.