Journal of Integrated
SCIENCE & TECHNOLOGY

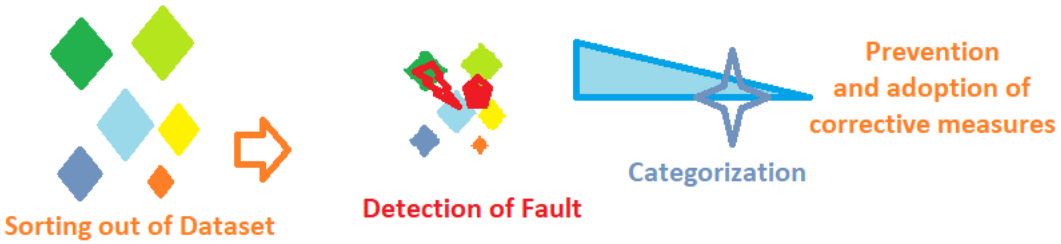
Failure recovery model in big data using the checkpoint approach

Sonika Chorey*, Neeraj Sahu

Computer Science and Engineering, G. H. Raisoni University, Amravati, India.

Received on: 12-Jan-2023 Accepted and Published on: 17-May-2023

ABSTRACT



Distributed Stream Processing systems are becoming an increasingly crucial aspect of Big Data processing platforms as customers grow ever more reliant on their capacity to deliver fast access to fresh findings. As a result, the ability of a system to tolerate failure is necessary for making prompt judgments based on these data. By using checkpoint, these systems typically achieve fault tolerance and the capacity to automatically recover from partial failures. An innovative method for automatic runtime optimization of fault detection and tolerance methods have been developed and used in this work.

Keywords: Big data, DSP, fault tolerance, Checkpoint, Deep learning

INTRODUCTION

The era of big data has arrived with the advent of the Internet of Things, cloud computing, biological science, and other associated new sectors.[1] Many data applications today require the processing of a huge volume of data in order to obtain insight into the data and resolve challenging issues. Data-intensive applications use parallel processing of massive amounts of data to get results quickly.[2] Distributed Stream Processing (DSP) systems are becoming an ever-more-important component of data processing environments due to the rising need to quickly process enormous volumes of unbounded data. Events must pass through this area of a graph of streaming operators in order to retrieve outcomes, which are most important as soon as the data arrive.[3]

A DSP can malfunction for a variety of reasons, including software bugs, hardware issues, user error, or even natural calamities. Hence, anytime a system or component failure occurs, it is essential to create an efficient system recovery mechanism so that the system may be reconfigured and restored in a timely way to restart the mission function.[4] In order to maintain the systems' efficacy, handling data replication in failure circumstances is essential. The key difficulties in data replication are in maintaining the replica's consistency while updates take place notwithstanding any failures that may occur while the transaction is being processed. By activating fault tolerance, these issues can only be resolved. In distributed computing, fault tolerance is a critical issue since. It maintains the transaction vulnerable to failure and in an operational state.[5] The ability of a system (computer, network, cloud cluster, etc.) to continue operating normally in the event that one or more of its components fail is known as fault tolerance.[6] The main goal of it is to keep the transaction going even if one of its components fails or malfunctions.[7] The dynamic method employed in DSP to maintain interconnected transactions while putting up with dependability and availability is known as fault tolerance.[8]

Checkpointing[9] has recently been added to boost fault-tolerance in the most recent actual deployments of distributed stream processing systems, such as Apache Flink and Storm[10], which typically display a directed acyclic graph (DAG) architecture of stream processing operators. Each map task's computing progress, including the current input data offset and computation results, can be periodically recorded using checkpointing. The most recent computing progress can be collected from the checkpoint in the event of task or node failure and the requirement to recover the failed task. The recovered job can then continue processing the

*Correspondence to: Sonika Anant Chorey
Tel: 9021899158
Email: sonikachorey@gmail.com

Cite as: *J. Integr. Sci. Technol., 2023, 11(4), 564.*

 ISSN: 2321-4635
http://pubs.thesciencein.org/jist

input data block starting from the most recent offset without having to process the entire data block.[11] There are various checkpointing methods that have been developed. Although checkpoints are necessary for an efficient recovery mechanism, the added overhead associated with checkpointing methods themselves can have a detrimental effect on system performance.[12] A few design elements that are prone to errors should be carefully taken into account. First off, saving all of these data greatly adds to the burden on computers. The subsequent high network bandwidth utilisation caused by storing intermediate findings on stable storage prevents computing nodes from exchanging data in a timely manner.[13] In order to lessen the detrimental consequences of the checkpoint method on system performance, several checkpoint placement policies, such as adaptive, age-dependent, or online checkpoints[14] have been investigated.

Globally, Check Point Software Technologies is a top supplier of cyber security solutions to businesses and governmental organisations. With an industry-leading catch rate for malware, ransomware, and other sorts of assaults, its solutions safeguard consumers from 5th generation cyberattacks. Check Point provides "Infinity" Complete Protection with Gen V advanced threat prevention, a layered security architecture that protects businesses' cloud, network, and mobile device-held data. The most complete and user-friendly single point of control security management system is offered by Check Point. Almost 100,000 enterprises of all sizes are protected by Check Point.[15]

Knowledge discovery in databases (KDD), another name for data mining, is a fast developing discipline. The demand for innovative methods to analyse, comprehend, or even display the vast amounts of stored data gathered from commercial and academic applications is what drives this technology. It is the process of extracting valuable information from massive amounts of data held in databases, data warehouses, or other information repositories, such as patterns, associations, changes, anomalies, and noteworthy structures. It can be utilised to assist businesses in reaching better decisions so they can continue to compete in the market.

Summarization, association, classification, prediction, and clustering are some of the main data mining functions that have been developed in both the commercial and research worlds.[5] A range of technologies, including database-oriented techniques, machine learning, and statistical techniques, can be used to achieve these functionalities. Applications and prototypes for data mining have recently been created for a variety of industries, including marketing, banking, finance, manufacturing, and health care. Several types of data, including time-series, geographic, telecommunications, web, and multimedia data, have also been subject to data mining applications. In general, the application domain and the type of data that are available have a significant impact on the data mining process, as well as the approach and function that will be used.

Structured Data Mining

The KDN website provides a list of the most popular data mining technologies that are either commercially or publicly available. The customer service database's structured data on sales, maintenance, and employee and customer information can be mined using these tools. It's intriguing to observe how many tools support various strategies, or different data mining methodologies. Darwin from Thinking Machine Corp., for instance, provides case-based reasoning, regression tree (CART), k-means algorithm, neural networks, and these functions for classification, prediction, and clustering. There are also certain tools that are solely intended to perform one particular data mining task. In order to get the best results, customers can choose from a variety of data mining methods for specific issue domains.[16]

Data Mining Method

The structure of the data mining process is shown in the figure below. It comprises of two main processes: the online fault diagnosis procedure and the offline knowledge extraction method. The first creates a knowledge base with the neural network models and a rule-base by extracting information from the customer service database. The second employs the four stages of the CBR cycle (retrieve, reuse, revise, and retain) to diagnose customer reported problems. The neural network models and the rule base cooperate inside the CBR cycle to support the second. It takes the user's description of the problem as input, maps it to the nearest fault-conditions of the faults already stored in the knowledge base, and then gets the appropriate checkpoint solutions for the user. The issue and its resolution are revised in light of user comments on the fault diagnosis procedure. At the end, the new finding is kept in mind as knowledge to improve future problem performance.[18]

LITERATURE REVIEW

Algorithm is a fault-tolerant scheduling. Advantage: Reduce projected execution time as much as possible and reduce time spent on fault-tolerant consumption. The performance increase ratio suffers from maximum probability failure, which is a drawback.[1] The approach uses a dynamic checkpointing policy-based evaluation mechanism for mission success probability (MSP). The advantage of using this approach is that optimisation problems are developed and addressed, and the best dynamic checkpoint policy, policy, or combination of checkpoint policy and element activation sequence is found to maximise MSP of the under consideration standby system.[2] A proactive method of system auto-tuning for Distributed Stream Processing jobs running on changing workloads is Phoebe. Phoebe can provide consistent end-to-end latencies. Moreover, it offers improved resource management. It disregards the recuperation time or the execution time.[3] Benefit of the Technique for Distributed Stream Processing Systems Checkpoint Intervals This model includes message latency, depth of the system topology, checkpoint interval, failure rate, checkpoint cost, failure detection, and restart costs, which improve the system's correctness and efficiency. This paradigm doesn't change to accommodate shifting workloads.[4] Quick Recovery Map Reduction is the method (FAR MR). In the event of task failure recovery, node failure recovery, and node failure recovery, improve computing job performance.[5] Disadvantage The procedure that uses the most resources is recovering from failures in tasks. Technique is McTAR (a Multi-trigger Checkpointing Tactic for fAstTAsk Recovery) McTAR shortens the job recovery time and boosts performance in failure-prone situations. The management of fail-stop faults, however, was not addressed, and the forecast based on random fault

information is unable to appropriately depict the condition of nodes.[6] This is known as Khaos method. The latency and recovery time violations are reduced by this Khaos model. Instead of producing an estimate, this model does not examine whether continuous optimisation is feasible.[7] Technique based on fuzzy logic. The system's reliability is improved by this model. Working with erroneous data causes the system's accuracy to decline.[8] Binary Vote Assignment on Grid (BVAG) method includes roll-back recovery and checkpoints. BVAGCRobtain a dependable performance of data replication in distributed databases with a failure situation. Data replication transactions take a long time to complete.[9]

Dynamic Check-Point Policy

Any operating element s(j) performs checkpointing activities during the mission so that the following checkpoint is carried out. after a predetermined number of operations j have been successfully completed since the previous checkpoint. It is expected that the system checkpointing process is completely accurate and error-free.[10]

We make j dependent on the index of the element being activated and the total number of operations left until the mission is finished in order to create a dynamic checkpoint policy.

There are K equal sections that make up the overall mission task (M operations). We suppose that k=xK/M portions should be finished if, following a failure, the remaining elements are required to carry out x actions in order to complete the mission. Upon failure, element s(j) shall activate and carry out n(j,k) equally spaced checkpoints until the task is complete, i.e. Checkpoint Dynamic Policy Dynamic Checkpoint Procedure Checkpoint Policy for Amic the completion of j=x/(n(j,k)+1) operations since the beginning.[11]

The checkpoint policy n(j,k)=k-1 for each j corresponds to j=M/K for any element activated at any moment, meaning the fixed checkpoint frequency with the sum of K-1 checkpoints throughout the operation.

According to the quantity and timing of element failures, the overall number of checkpoints carried out during the mission may vary. The dynamic checkpoint policy is defined by the specified value of K and the function n(j,k) for j=1,...,N and k=1,...,K.
Using K and n(j,k), one can specify the number of checkpoints that element s(j) should complete given that it must carry out x mission activities till the completion of the mission.

In general, the mission's total number of checkpoints might vary depending on the quantity and timing of
elemental blunders.The dynamic checkpoint policy is defined by the specified value of K and the function n(j,k) for j=1,...,N and k=1,...,K.

The number of checkpoints that element s(j) should conduct given that it must complete x mission activities is defined as nj(x)=n(j,xK/M) when K and n(j,k) are available. Because element s(1) is always activated at the start of the mission and only n1(M) specifies the number of checkpoints this element completes, it is important to note that for j=1, any value of n(1,k) for kK has no meaning.[17]

Take, for instance, a work finished by element s(4) in Figure 1 after three failures of the earlier elements. The checkpoint policy has the following parameters: K=3, n(1,3)=4, n(2,1)=0, n(2,2)=2, n(2,3)=1, n(3,1)=1, n(3,2)=n(3,3)=2, n(4,1)=1, n(4,2)=n(4,3)=2, and n(4,3)=0. Mission operations and checkpoint operations are differentiated in this section and throughout the text. Checkpoint operations are not counted in the number of mission operations remaining until the mission is finished. In particular, n(1,3)=4 establishes that element s(1) performs 4 checkpoints. In other words, if the mission job is not failed, element s(1) should perform checkpoints after finishing each 1/5 of the assignment, i.e., 1=M/5.[18]

The element s(1) in this illustration fails between the first and second checkpoints.

After retrieving the previously saved information, the remaining components should carry out x=M-1=4M/5 mission actions to finish the task. k=xK/M=(4M/5)(3/M)=3 following the failure of element s(1). Assuming that n(2,3)=1, element s(2) should perform checkpoints when each remaining mission job is half-complete, or 2=(4M/5)/2=0.4M. Before reaching the end of its first checkpoint, element s(2) fails.

The amount of work that has to be done after this failure remains the same because no more successful checkpoints were reached.[18,19]

Hence, x=4M/5 and k=3. Due to the fact that n(3,3)=2, element s(3) must conduct checkpoints after finishing each 1/3 of the mission's remaining tasks, or 3=(4M/5)/3=4M/15. Between the first and second checkpoints of element s(3), it fails. After data retrieval, the remaining components should finish x=4M/5- 3=8M/15 mission activities, which translates to k=(8M/15)(3/M)=2. Element s(4) shall perform checkpoints after finishing each 1/3 of the remaining mission job, or 4=(8M/15)/3=8M/45, since n(4,2)=2 and element s(4). Element s(j) performs the h-th checkpoint procedure, which calls for Bjh operations. Depending on the checkpoint scheme chosen, Bjh may be a result of the work completed since the last successful checkpoint or from the mission's inception.

The quantity of data saved during the checkpoint, and therefore Bjh, rely on the amount of work finished since the last successful checkpoint, specifically if an incremental checkpoint technique is utilised. In this instance, Bjh=b(j) for every h=1,..., n(j,k) is true, and b(x) is a function that returns the number of operations required to store the data produced during x mission activities. Bjh relies on the number of mission operations completed from the start of the mission till the checkpoint time if a total checkpoint technique is used.[10] Large-scale use of big data and the development of Internet of Things (IoT) technologies have made it possible for cities to gain insightful knowledge from a wealth of data that is produced in real-time. Numerous IoT devices in a smart city continuously produce data, which must be analysed quickly using big data techniques. The applicability of using distributed stream processing frameworks at the data processing layer of a smart city is examined in this article, along with the acceptance and maturity of these frameworks across various Smart City use cases. The goal of our investigations is to compare the effectiveness of three SDPSs: Apache Storm, Apache Spark Streaming, and Apache Flink.[14] In this research, consider a novel stochastic model for the file recovery action with checkpointing under a homogeneous Poisson failure of the system. the current checkpoint model heavily depends on the system age. In order to find the best checkpoint interval that

maximises system availability while taking into consideration the queuing effect caused by inactivity in the transaction processing system, we offer three different types of approximation approaches. The checkpoint model, which is based on three approximation approaches, is quantitatively compared with older models using numerical examples, and it is demonstrated that it can reduce system overhead that may result in unanticipated system downtime.[26]

In this research, when a system failure happens based on a homogeneous Poisson process, we consider a new stochastic model for file recovery action with checkpointing. In contrast to the models by Gelenbe (1979) and Goes and Sumita (1995), the current checkpoint model heavily depends on the system age. In order to find the best checkpoint interval that maximises system availability while taking into consideration the queuing effect caused by inactivity in the transaction processing system, we offer three different types of approximation approaches. The checkpoint model, which is based on three approximation approaches, is quantitatively compared with older models using numerical examples, and it is demonstrated that it can reduce system overhead that may result in unanticipated system downtime.[27]

A prominent method for speeding up computer system recovery from errors is checkpointing. Checkpointing enables one to lessen the processing time loss brought on by faults by preserving interim states of programmes in a dependable storage medium. The duration of the checkpoint breaks has an impact on how quickly the programmes run. Long checkpoint intervals result in lengthy reprocessing times, while frequent checkpoints have a large overhead. In this research, we describe an online checkpoint placement technique. When deciding whether or not to put a checkpoint, the algorithm leverages online knowledge of the current cost of a checkpoint. We demonstrate how to evaluate a program's execution time using this approach.[28]

For broad rollback and recovery systems, a numerical method for calculating the best dynamic checkpointing strategies is described. Modelled as a Markov renewal decision process, the system. Reprocessing-dependent recovery times, general failure distributions, and random checkpointing intervals are all acceptable. To maximise the average system availability across an unlimited time horizon, a dynamic decision rule must be found. It is suggested to use computational methods to approximate such a regulation. This method is based on stochastic dynamic programming with value and policy functions approximated using splines or finite elements. Illustrations using numbers are offered.[29] In-depth research has been done over the past 20 years to develop effective checkpointing methods for use in conventional distributed computing. More focus has recently been placed on developing checkpointing methods for mobile systems. Some of these protocols were developed specifically for mobile systems, while others were modified from the conventional distributed environment. A checkpoint is a predetermined moment in a programme where regular processing is paused particularly to preserve the status data required to enable processing to resume at a later time. The procedure of checkpointing involves preserving the status data. This study reviews the checkpointing techniques for mobile distributed systems that have been published in the literature.[30]

Another study simulates a mission that requires repairable computing equipment to complete a certain amount of work within a given amount of time or by a certain deadline. To enable an efficient system recovery and prevent starting the mission over from scratch when a system failure occurs, the system is subjected to a series of complete and incremental data backup procedures while the mission is in progress. The system time-to-failure can follow any arbitrary form of distributions, however the repair time is fixed. In order to assess the mission success probability and anticipated completion time of the studied repairable real-time computing systems subject to mixed full and incremental backups, a new numerical algorithm was initially developed in this study. Validity of the suggested evaluation algorithm have been reported.[31]

CHALLENGES

• Allocating sufficient resources to ensure recovery is within acceptable bounds in the presence of QoS recovery time targets is one of the difficult challenges.[3]

• In a streaming application, the depth of the system topology and message delay have an impact on the multi-level checkpointing procedure.[4]

• Depth of the system topology and message latency • The majority of the fault tolerance strategies now in use are ineffective and recovery from failures takes an excessively lengthy time.[5]

• The depth of the system's topology and message latency. Keeping interim results on reliable storage also uses up a lot of the network's capacity, making it difficult for compute nodes to exchange data promptly.[7]

• The main problems encountered in coordinated check points include deep system topology, communication delay, consistent checkpoint, and high latency for saving the checkpoints storage.

OBJECTIVES

• To analyze and explore various failure detection and fault tolerance methods in big data application

• To design and develop failure detection model based on modified Deep learning techniques.

• To design and develop a fault tolerance model based on the deep learning techniques.

• To design and develop a novel algorithm for the effective tuning of hyper parameters in classifiers

• To design and develop an automated notification for system failure, which enables to take backup up to current check point before the destruction of data.

Problem Statement

Modern multi-core CPUs are impressively capable of handling sophisticated, huge transactions in databases. Depending on the magnitude of the transaction, these databases can process millions to billions of transactions every second. Such databases must, however, be resilient to errors. Logging to disc can be used to ensure database persistence. Database systems also employ frequent checkpoints in addition to logging, as this speeds up recovery and prevents the system from running out of disc space.

There are various difficulties in implementing persistence for a quick and big in-memory database. Logging is quick because it makes full use of the disc throughput by using sequential writes. Checkpoints demand a tour over the full database, though. Database performance may be impacted by data transfer, cache pollution, and logging interference brought on by writing the database to disc. It is difficult for such a system to recover.

More than 50 GB of log data can be produced per minute by modern databases, which can process tens of millions of tiny transactions every second. This is up to many orders of magnitude higher than the results reported in earlier studies on in-database durability in terms of transaction rates and log sizes. Recovery is tough since there is so much data to read and recover when the log size is so large.

There is a need ot a plan and its execution to address the problems listed above, especially as follows:

• Making all of the significant durability mechanisms parallel is possible and recommended.

• Logging and checkpointing in tandem can be quick without affecting typical transaction processing.

• During recovery, log replay becomes a bottleneck because of the large flow of data.

Depth of the system topology and message delay Distributed stream processing (DSP) systems are becoming an increasingly crucial component of data processing environments due to the growing need to quickly process enormous volumes of unbounded data. The DSP frequently fails for a variety of reasons, including software faults, hardware issues, user error, or natural calamities. So that the system may be quickly changed and restored, it is essential to create an efficient system recovery mechanism. Several checkpoint strategies have been proposed for failure recovery, but these models have added overhead. So, the goal of this project is to provide

a) A defect or failure detection technique based on deep learning that uses checkpoints and has lower computing overheads.

b) To fully satisfy the QoS specifications, which include fast throughput and minimal delay.

METHODOLOGY

recovery model. Users enter a certain amount of tasks into the cloud centre to begin the technique. On the basis of internal schedule management, the tasks are given to the virtual machine. The adjusted deep learning system's required parameters will be extracted within a predetermined time frame using the monitoring component. The fault detection system will forecast the presence or absence of a fault based on the input parameters that this component retrieves from the Deep CNN-based fault detection system. The suggested Advanced rescue optimisation, which will be created by fusing the features of teaching and learning with those of human rescuers,[21,22] will be used to modify the hyperparameters of the deep learning classifier. The monitoring component once more retrieves the parameters needed by the Deep CNN based checkpoint-enabled tolerance mechanism in the case of a fault. Lastly, the fault tolerance system assigns the proper checkpoints through the redundant output based on the built-in fault tolerance algorithms.

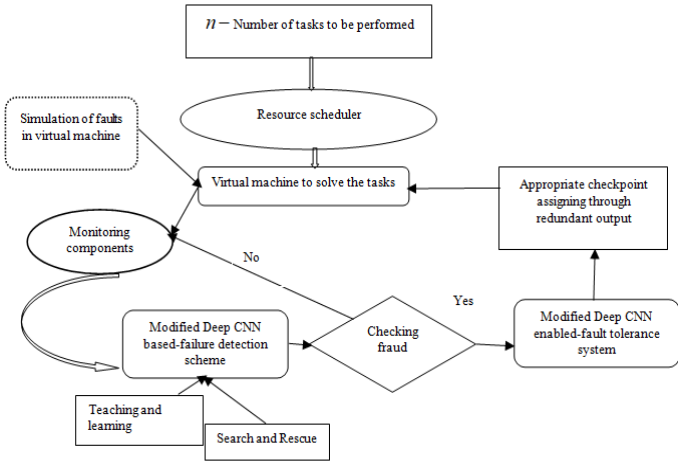


Figure 1. Flow chart for the adopted Methodology

Performance metrics: Latency, recovery time and average percentage of error.

RESULTS AND DISCUSSION

This section implements the performance and comparative analysis for the partial recovery system's deep CNN approach. Accuracy, recovery time, precision, and recall are the parameters that have been taken into account when assessing the performance of current CNN model.

Performance analysis using database

In the CNN performance is analyzed by considering the training percentage, the accuracy of the MRRO-deep CNN at the initial breaking epoch 4 and the final breaking epoch 24 for the training percentage 90 is 93.500 % and 100.200 %, . The recovery time of the MRRO-deep CNN at the initial breaking epoch 4 and the final breaking epoch 24 for the training percentage 90 is 19.050 s and 9.200 s, CNN at the initial breaking epoch 4 and the final breaking epoch 24 for the training percentage 90 is 93.052 % and 98.050 %. The recall of CNN for the training percentage 90 at the initial breaking epoch 4 and the final breaking epoch 24 is 72.95% and 76.81%.

Comparative discussion

Methods	90 % training			
	Database			
	Accuracy %	Recovery time (s)	Precision %	Recall %
Fault tolerant scheduling algorithm	84.016	10.700	70.500	72.400
FAR-MR	88.275	11.326	72.080	73.995
ECAC	91.000	9.100	72.092	75.692
Deep CNN	93.500	9.200	72.956	76.817

CONCLUSION

In this study, we developed a checkpoint-based failure recovery model built on a modified Deep CNN. In order to maintain the lowest latency and avoid recovery time violations, it monitors the

virtual machine to identify task execution errors. Due to the proper tuning of the Deep CNN classifier, it is anticipated that the developed Deep CNN based failure recovery model employing checkpoint strategy would outperform the current methods.

CONFLICT OF INTEREST

Authors do not have any conflict of interest in publishing of this work. No Academic or financial interest to be declared for this work.

REFERENCES AND NOTES

1. H. Xuan, S. Wei, W. Tong, D. Liu, C. Qi. Fault-Tolerant Scheduling Algorithm with Re-Allocation for Divisible Task. *IEEE Access* **2018**, 6, 73147–73157.
2. G. Levitin, L. Xing, Y. Dai, V.M. Vokkarane. Dynamic Checkpointing Policy in Heterogeneous Real-Time Standby Systems. *IEEE Trans. Comput.* **2017**, 66 (8), 1449–1456.
3. M.K. Geldenhuys, D. Scheinert, O. Kao, L. Thamsen. Phoebe: QoS-Aware Distributed Stream Processing through Anticipating Dynamic Workloads. *Proceedings - IEEE International Conference on Web Services, ICWS 2022*. 2022, pp 198–207.
4. S. Jayasekara, A. Harwood, S. Karunasekera. A utilization model for optimization of checkpoint intervals in distributed stream processing systems. *Futur. Gener. Comput. Syst.* **2020**, 110, 68–79.
5. Y. Zhu, J. Samsudin, R. Kanagavelu, et al. Fast Recovery MapReduce (FAR-MR) to accelerate failure recovery in big data applications. *J. Supercomput.* **2020**, 76 (5), 3572–3588.
6. Y. Fang, Q. Chen, N. Xiong. A multi-factor monitoring fault tolerance model based on a GPU cluster for big data processing. *Inf. Sci. (Ny).* **2019**, 496, 300–316.
7. A multi-trigger checkpointing tactic for fast task recovery in mapreduce". *IEEE Trans. Serv. Comput.*
8. M.K. Geldenhuys, B.J.J. Pfister, D. Scheinert, L. Thamsen, O. Kao. Khaos: Dynamically Optimizing Checkpointing for Dependable Distributed Stream Processing. *Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS 2022*. 2022, pp 553–561.
9. N. Hagshenas, M. Mojarad, H. Arfaeinia. A Fuzzy Approach to Fault Tolerant in Cloud using the Checkpoint Migration Technique. *Int. J. Intell. Syst. Appl.* **2022**, 14 (3), 18–26.
10. S. Hafizah Sy Ahmad Ubaidillah, B. Alkazemi, A. Noraziah. An Efficient Data Replication Technique with Fault Tolerance Approach using BVAG with Checkpoint and Rollback-Recovery. *Int. J. Adv. Comput. Sci. Appl.* **2021**, 12 (1), 473–480.
11. J. Mervis. Agencies Rally to Tackle Big Data. *Science*. 2012, pp 22–22.
12. G. Cattaneo, U.F. Petrillo, R. Giancarlo, G. Roscigno. An effective extension of the applicability of alignment-free biological sequence comparison algorithms with Hadoop. *J. Supercomput.* **2017**, 73 (4), 1467–1483.
13. Y. Zhu, S. Juniarto, H. Shi, J. Wang. VH-DSI: Speeding up data visualization via a heterogeneous distributed storage infrastructure. In *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*; **2016**; Vol. 2016-January, pp 658–665.
14. H. Nasiri, S. Nasehi, M. Goudarzi. A survey of distributed stream processing systems for smart city data analytics. In *ACM International Conference Proceeding Series*; **2018**; pp 1–7,.
15. H. Isah, T. Abughofa, S. Mahfuz, et al. A survey of distributed data stream processing frameworks. *IEEE Access* **2019**, 7, 154300–154316.
16. H. Nasiri, S. Nasehi, M. Goudarzi. Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities. *J. Big Data* **2019**, 6 (1), 1–24,.
17. A. Sarı, E. Çağlar. Performance Simulation of Gossip Relay Protocol in Multi-Hop Wireless Networks. In *Owner: Girne American University.*; **2015**; p 145.
18. P. Carbone, A. Katsifodimos, S. Ewen, et al. Apache flink: Stream and batch processing in a single engine. *Bull. Tech. Comm. Data Eng.* **2015**, 38 (4), 28–38.
19. P. Carboney, S. Ewenz, G. Fóra, et al. State management in Apache Flink: ® consistent stateful distributed stream processing. *Proc. VLDB Endow.* **2017**, 10 (12), 1718–1729.
20. E. Kail, K. Karóczkai, P. Kacsuk, M. Kozlovszky. Provenance based checkpointing method for dynamic health care smart system. *Scalable Comput.* **2016**, 17 (2), 143–153.
21. A. Shabani, B. Asgarian, M. Salido, S. Asil Gharebaghi. Search and rescue optimization algorithm: A new optimization method for solving constrained engineering optimization problems. *Expert Syst. Appl.* **2020**, 161, 113698.
22. R. V. Rao, V.J. Savsani, D.P. Vakharia. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. In *CAD Computer Aided Design*; **2011**; Vol. 43, pp 303–315.
23. P.A. Chorey, N. Sahu. Secure Online Transactions Using a Blockchain-Based Checkpoint Approach. *Comput. Integr. Manuf. Syst.* **2022**, 28 (11), 1–11.
24. A.W. Burange, V.M. Deshmukh. Trust based secured Routing System for low power networks. *J. Integr. Sci. Technol.* **2023**, 11 (1), 431.
25. S. Muhammad Abrar Akber, H. Chen, Y. Wang, H. Jin. Minimizing Overheads of Checkpoints in Distributed Stream Processing Systems. In *Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking, CloudNet 2018*; **2018**; pp 1–4.
26. Y. Zhang, K. Chakrabarty. Adaptive Checkpointing with Dynamic Voltage Scaling in Embedded Real-Time Systems. *Embed. Softw. SoC* **2005**, 449–463.
27. T. Dohi, N. Kaio, K.S. Trivedi. Availability models with age-dependent checkpointing. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems*; **2002**; pp 130–139.
28. A. Ziv, J. Bruck. An on-line algorithm for checkpoint placement. In *IEEE Transactions on Computers*; White Plains, NY, **1997**; Vol. 46, pp 976–985.
29. J. Malenfant. Computing Optimal Checkpointing Strategies for Rollback and Recovery Systems. *IEEE Trans. Comput.* **1988**, 37 (4), 491–496.
30. A. Khunteta, P. Kumar. An Analysis of Checkpointing Algorithms for Distributed Mobile Systems. *Int. J. Comput. Sci. Eng.* **2010**, 02 (04), 1314–1326.
31. G. Levitin, L. Xing, Q. Zhai, Y. Dai. Optimization of Full versus Incremental Periodic Backup Policy. *IEEE Trans. Dependable Secur. Comput.* **2016**, 13 (6), 644–656.