Ain Shams Engineering Journal 13 (2022) 101538

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Fault tolerance in big data storage and processing systems: A review on challenges and solutions

Muntadher Saadoon, Siti Hafizah Ab. Hamid *, Hazrina Sofian, Hamza H.M. Altarturi, Zati Hakim Azizul, Nur Nasuha

Department of Software Engineering, Faculty of Computer Science and Information Technology, Universiti Malaya, 50603 Kuala Lumpur, Malaysia

ARTICLE INFO

Article history:
Received 5 December 2020
Revised 21 February 2021
Accepted 10 June 2021
Available online 14 July 2021

Keywords:
Fault tolerance
Fault detection
Fault recovery
Big data storage
Big data processing

ABSTRACT

Big data systems are sufficiently stable to store and process a massive volume of rapidly changing data. However, big data systems are composed of large-scale hardware resources that make their subspecies easily fail. Fault tolerance is the main property of such systems because it maintains availability, reliability, and constant performance during faults. Achieving an efficient fault tolerance solution in a big data system is challenging because fault tolerance must meet some constraints related to the system performance and resource consumption. This study aims to provide a consistent understanding of fault tolerance in big data systems and highlights common challenges that hinder the improvement in fault tolerance efficiency. The fault tolerance solutions applied by previous studies intended to address the identified challenges are reviewed. The paper also presents a perceptive discussion of the findings derived from previous studies and proposes a list of future directions to address the fault tolerance challenges.

1. Introduction

Big data systems gain considerable attention from the research community and the industry due to the growing volume and importance of data [1]. Sources, such as Internet of Things sensors, social media, and healthcare applications, increasingly generate massive amounts of data. For example, the log production rate of social media applications can reach several terabytes or petabytes per day [2]. The International Data Corporation predicts that approximately 163 zettabytes of data will emerge by 2025 [3]. Traditional Relational Database Management System (RDBMS), such as MySQL [4], cannot handle the increased volume of data because of hardware limitations on a single server. Data storage and processing must be distributed on reliable servers to transfer massive data into the desired form. The possibility of failures increases when the number of servers increases [5]; thus, decision-makers and developers regard fault tolerance as an important property of big data systems because it enables a failure-free execution and prevents the degradation of performance. Accordingly, popular big data frameworks, such as MapReduce [6] and its open-source implementation Apache Hadoop [7] have considered fault tolerance by offering various fault-tolerance approaches such as data redundancy, checkpointing, and speculative execution that enables resiliency against failures. However, they cannot always fulfil their reliability [8] and performance requirements because failures become a norm and are no longer exceptions in large-scale environments [9].

Fault tolerance is defined as the property of a system that continues to operate even during faults [10,11]. Over the years, the demand for achieving efficient fault-tolerant solutions has been increasing [12–20] to improve the reliability and performance of big data systems [21]. This demand corresponds to the increase in the number of software and hardware failures caused by scalability, complexity, and interdependency of the underlying environment resources. A large-scale cluster consisting of 1000 super reliable servers with a mean time between failures of 30 years is expected to have one failure per day [5]. More than 1000 individual nodes and hard drive failures affect the cluster during its first year

* Corresponding author.
E-mail addresses: core93@yahoo.com (M. Saadoon), sitihafizah@um.edu.my (S.H. Ab. Hamid), hazrina@um.edu.my (H. Sofian), altarturih@gmail.com (H.H.M. Altarturi), zati@um.edu.my (Z.H. Azizul), nasuha@um.edu.my (N. Nasuha).
Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.asej.2021.06.024
2090-4479/

of service [22,23]. The recovery time of these failures can approximately reach 2 days [24], and this condition extremely violates the performance.

On this basis, an efficient fault-tolerant solution for big data systems is necessary for achieving better availability, reliability, and performance in the presence of faults. Consequently, a literature review is important to understand the mechanism by which fault tolerance has been addressed in big data systems and to gain an insight on challenges and recent solutions in this field towards new promising research directions. To date, review articles concentrating on the challenges and solutions of fault tolerance in big data systems have been scarce. To the best of our knowledge, only Memishi et al. [25] focused on optimization mechanisms of fault tolerance from 2004 to 2016 in the MapReduce systems. The current work strives to review and examine the previous studies involving fault-tolerant solutions for big data systems for addressing this gap. Specifically, this work focuses on the challenges faced by the previous researchers and their proposed solutions to overcome the challenges.

The remainder of the paper is organised as follows. Section 2 provides an overview of big data systems including the data storage and processing systems. Section 3 provides an overview of fault types and fault tolerance approaches. Section 4 generally classifies the fault tolerance challenges and reviews the solutions noted by the previous studies, while Section 5 discusses and analyzes the findings of the proposed solutions. Section 6 suggests some future research directions in this area and our conclusion.

2. Big data systems

Big data includes data, data storage, data processing, data analysis, information management, interfaces, and visualization [26]. Among them, data storage and processing crucially require fault tolerance as they manage the storage and computational resources for big data applications. These resources are also vulnerably exposed to failures.

2.1. Storage system

The existing storage technologies have been restricted to store and manage data due to the vast volume of the generated data. RDBMSs are traditionally used to manage structural data [27]. However, these systems cannot store and manage big data. A scalable and reliable storage architecture that can achieve high data availability in a distributed manner is required to manage the huge volume of data. Several storage systems, such as Google File System (GFS) [28], Hadoop Distributed File System (HDFS) [29] and OpenStack Swift [30], have been proposed to handle the big data storage challenges. These systems consist of distributed storage devices connected through the network and support virtualisation, distribution, and scalability to efficiently fit with a massive volume of data. Distributed storage devices are typically a network with attached storage and virtualisation ability [31]. Storage virtualisation is a technique that presents a logical view of the physical storage resources as a signal storage pool. The network of the storage devices is used to access the stored information despite their locations or modes. The storage system can have three file system modes; namely, file storage, block storage, and object storage, which are used in big data storage systems.

- *File Storage*: Data are organised hierarchically in files and the storage system stores all the file information as metadata. The files are reachable by determining the path to a specific file stored in the metadata.
- *Block Storage*: Data are divided into blocks, and each block holds a part of the data. The storage system assigns a unique ID to each block to allow application access and combine the blocks by their IDs.
- *Object Storage*: Data are encapsulated with metadata in objects. Each object has unique metadata with certain configurations, such as geographical location, replica number, and protection level.

O'Reilly [32] provided a further discussion on the advantages and drawbacks of each file system mode. GFS and HDFS are the widely used storage systems and a comparison between them is presented by Verma and Pandey [33].

2.2. Processing system

With the massive increase in data and the advancement in big data storage technologies for handling data storage and management in distributed systems, big data processing systems have been proposed to transform the massive amount of data into usable and desirable forms to facilitate the development of scalable solutions and algorithms. For example, data scientists use big data processing systems for data-intensive text processing, assembly of large genomes, machine learning and data mining, and large-scale social network analysis. The widely used big data processing systems are Hadoop [7], Spark [34], Storm [35], Samza [36], and Flink [37], and they are open-source projects under the Apache Software Foundation [38]. These systems contain three main layers: processing engine, cluster manager, and processing framework [39].

- *Processing Engine*: An abstraction that allows performing simple computations whilst hiding the details of parallelisation, distribution, load balancing, and enabling fault tolerance.
- *Cluster Manager*: A centralised service for maintaining the cluster configuration. It ensures dynamic resource sharing and provides efficient resource management for distributed data processing frameworks.
- *Processing Framework*: A framework that consists of a set of tools that allows efficient analytics of a massive volume of data. Such a framework can support several application programming interfaces (APIs) to integrate other types of applications and algorithms, such as scalable machine learning algorithms, graph-parallel computation, and interactive analytics or streaming,

Big data processing systems offer different features that can fit in various big data use cases. For example, Hadoop supports batch processing, and Storm and Samza support stream processing. By contrast, Spark and Flink can be used for batch and streaming. Batch processing is used when the data are collected or stored in large files. The processing result, such as index generation from documents for internet-scale search, is required when the processing is completed. In the meantime, stream processing is used when the data are continuous and must be rapidly processed, such as the analysis of users' Tweets posted on Twitter. Inoubli et al. [40] provided an experimental survey that includes a comparative study of the respective big data processing systems.

3. Fault tolerance

3.1. Fault types

Fault, error, and failure are fundamental concepts in fault tolerance [41,42]. These concepts are due to different conditions that could be either caused by software or hardware resources, and

they have varied effects on the system behaviour. The relationship among fault, error, and failure from the hardware to the software level, and vice versa, is conceptually illustrated in Fig. 1. A fault is an unusual hardware or software condition that produces an error when the fault is active. An inactive fault refers to a condition in which the fault does not produce an error because it is out of the boundaries of the system functionality. Once an error is produced by an active fault, the error leads to a deviation of the expected logical value, which results in a failure. Failure means the inability of the system to perform its intended function. Examples of frequent fault sources are processor or memory damage, network overflow, and corrupted storage devices. Mistakes in software specifications or implementation and external misuses, such as unanticipated inputs, can also generate faults in the system.

Faults can be generally classified into two types: *permanent* and *transient*. A *permanent* fault, which is also called a fail-stop [43], is continuous and remains for a fixed period. This type of fault is easy to detect and is localised because the faulty component remains unresponsive after a permanent fault occurs [44]. By contrast, a *transient* fault, which is also known as a straggler [18] or a fail-stutter [45], makes a system accessible but with a poor performance [46]. The nature of the transient fault is slightly different from that of a permanent fault because a transient fault occurs at a random frequency; hence, it is more difficult to be detected than the permanent ones.

3.2. Fault tolerance approaches

Fault tolerance is the property of a system that maintains continuous running of service even during faults. Fault-tolerant systems are built on two main key concepts: fault detection and recovery [44]. These two concepts can be achieved based on various fault-tolerance approaches as classified in Fig. 2.

Fault detection is the first building block in a fault-tolerant system that enables detecting faults as soon as they manifest themselves within the system. Heartbeat detection and fault prediction are stable fault detection approaches used in large-scale systems. The heartbeat approach is designed based on an explicit and periodic exchange of heartbeat messages between two components during error-free periods [44]. A sample example is as follows: if component A has not received the heartbeat message from another associated component B within a specified time frame, then A would declare that B has failed, and the fault-tolerant system will be ready to apply a treatment action for recovery. This approach has been implemented in numerous large-scale systems such as HDFS, YARN, and Strom. A faulty component can only be detected when it literally fails to send/receive heartbeat messages. In contrast, detecting faults before they occur can be addressed with the fault prediction approach. Fault prediction predicts possible upcoming faults using a statistical model and historical information of previously executed workloads. The statistical model is used to inspect the workload's dependencies and parameters such as execution time, scheduling constraints, resource usage, and machine workload. Soualhia et al. [47] collected historical workloads of Google cloud cluster over a period of one month and achieved precision up to 97.4%, and a recall of up to 96.2% in predicting faults based on the Random Forest algorithm. Further, this prediction model has been adopted in Hadoop to improve the scheduling decision for efficient fault recovery before the existence of fault; hence, it enhances the overall reliability and performance of Hadoop [24].

On the other hand, fault recovery is employed to return the faulty component to its normal behaviour after detecting a fault. In the storage systems, data redundancy based on replication and/or erasure coding approaches is used to ensure data availability and reliability. The replication approach simply works by creating multiple copies of the original data and storing them on different disks for availability and fault tolerance [48]. This approach has been employed in GFS, HDFS, RAMCloud [49], and Windows Azure Storage (WAS) [50] to provide high data availability. With the replication, data are replicated on multiple servers and different racks once the data are uploaded to the storage system. The storage system has information about replicas and their locations in the metadata file. During a fault, the storage system refers to the replicated data directory for recovery when the original data are inaccessible. In the worst case, replication is futile if the original data and the replica are lost. Another popular data redundancy approach is erasure coding. Erasure coding creates and stores parity data along with the original data on another disk instead of replicating the entire chunks of data. In case of data loss on one disk due to a fault, parity data can be reconstructed to produce the original data. In (n, k) erasure coding, the data of size A is chunked into k equal chunks, and the parity data are represented as n – k. Therefore, any k out of n would reconstruct the original data. For instance, any two inaccessible chunks of data can be reconstructed from the parity data if the erasure coding is represented as (4, 2). The two types of erasure coding; namely, maximum distance separable (MDS) and non-MDS, are commonly used. The MDS erasure-coded system can reconstruct any lost chunk of data based on the parity data, while the non-MDS can only reconstruct a few of the lost chunks [32]. Erasure coding has recently been implemented in HDFS [51] due to its storage efficiency. Besides, a popular implementation of erasure coding is the Reed–Solomon coding (RSC), which is used by Facebook and the Microsoft storage systems [43,52].

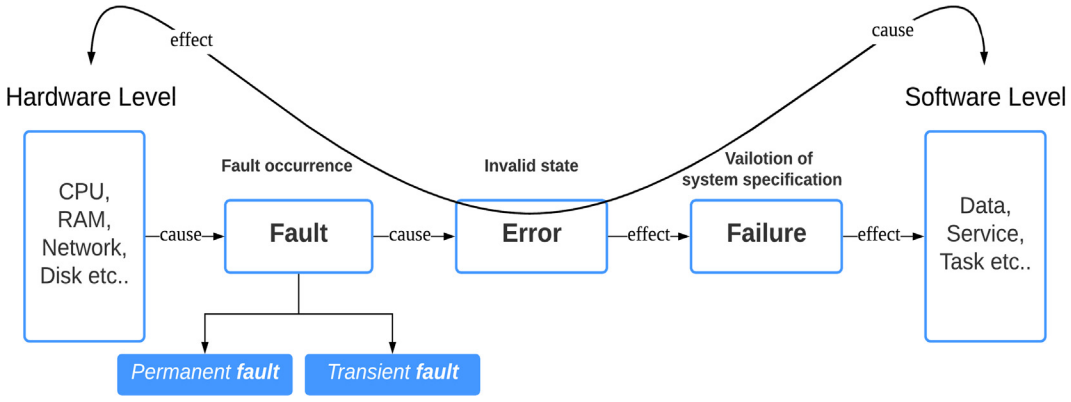


Fig. 1. Relationship among fault, error, and failure from the hardware to the software level, and vice versa.

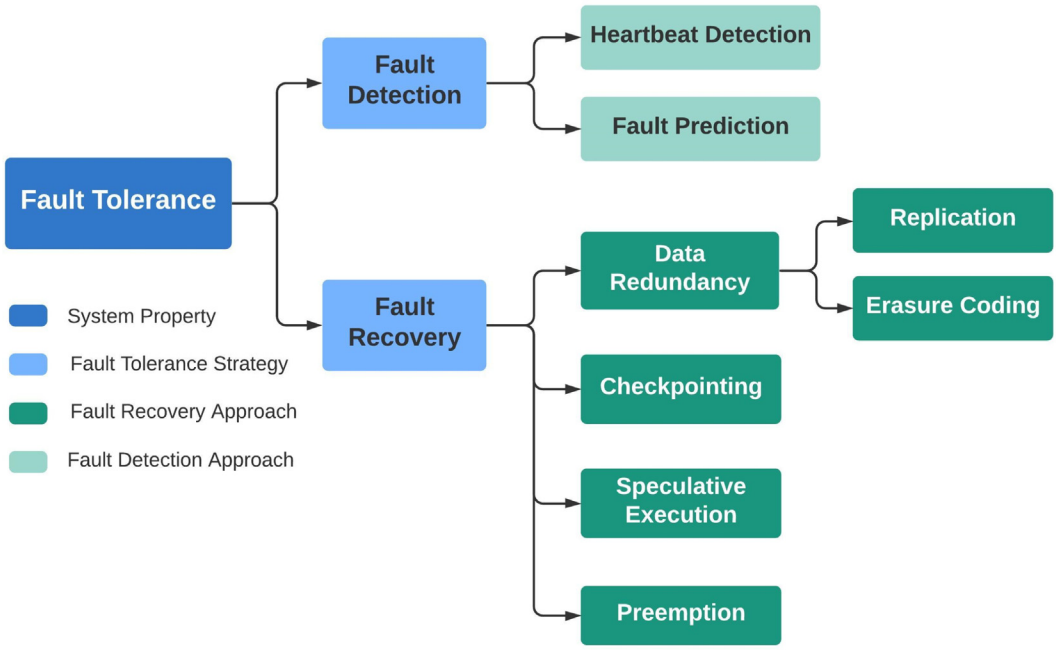


Fig. 2. Classification of fault tolerance approaches.

Checkpointing, speculative execution and preemption are also common fault tolerance approaches used in large-scale data processing systems where the fault occurs at the processing time. Checkpointing works by recording the state of an active node/process to store it on another standby node. When the node fails, the standby node takes over on the latest recorded state of the active node for an efficient and fast fault recovery. Checkpointing is normally used in critical systems that involve real-time transactions or stream processing where the latency is very low. For example, Flink utilizes checkpointing to achieve exactly-once-processing guarantees by taking a snapshot of the operator state, including the current position of the input streams at regular intervals [53]. Next, speculative execution duplicates the active tasks when their performance is lower than a certain threshold because they are suspected to fail. This approach allows a faster fault recovery by considering the output of the high-performing tasks and terminating their identical tasks that are performing poorly. YARN uses this approach for fault recovery. Preemption is also a fault recovery approach that is usually used by the task scheduler to provide efficient fault recovery when the cluster runs out of full resource capacity. It requires a predefined task preemption policy to terminate the low-priority tasks and allow the available slots for re-executing the high-priority tasks if they fail.

4. Fault tolerance challenges and solutions in big data systems

The recent challenges of fault tolerance in big data storage and processing systems can be generally classified into storage capacity, disk I/O, and bandwidth usage, fault detection latency, and fault recovery efficiency as highlighted in Fig. 3.

4.1. Challenges and proposed solutions for the storage system

Fault tolerance strategies in the storage system are achieved using data redundancy approaches, including replication and erasure coding. In replication, the system reliability is directly associated with the storage overhead. Thus, improving the storage overhead without sacrificing reliability is a great challenge. Although erasure coding offers huge storage savings with fair reliability compared with replication, it introduces high disk I/O consumption and data transfer overhead [54]. These challenges are observed in most literature when conducting fault tolerance in big data storage systems.

4.1.1. Storage capacity

The storage systems currently use the replication approach to ensure data availability and reliability. This approach generates a high level of data redundancy. For instance, GFS and HDFS proactively create by default three replicas once the data are uploaded to the storage system. Specifically, storing 1 TB of data would need 3 TB of storage space, which costs a huge amount of storage space and increases the level of energy consumption.

4.1.2. Disk I/O and bandwidth usage

Erasure coding attracts attention nowadays due to its ability to offer efficient storage consumption in data redundancy with promising reliability assurance compared with replication [31]. However, the reconstruction process of the parity data stored on multiple disks requires more disk I/O operations than replication [52,55]. This challenge increases the latency and the number of the read requests when accessing the parity data; it also increases the data transfer among the nodes, especially if data are stored in the distributed disks that are hosted on several nodes with different locations.

4.1.3. Proposed solutions

Works by Li et al. [19,56] and Wei et al. [57] propose solutions on minimising the replicated data while at the same time maintaining data reliability requirements by focusing on the relationship between the number of the replicated data and the requirements of data availability and reliability. Works by Long et al. [58] and Hassan et al. [59] use multi-objective strategies to overcome the storage capacity overhead. Works by Huang et al. [52,60] and Sathiamoorthy et al. [43] focus on improving erasure code by manipulating local parity.

Wei et al. [57] proposed the cost-effective dynamic replication management (CDRM) scheme for achieving a minimal number of

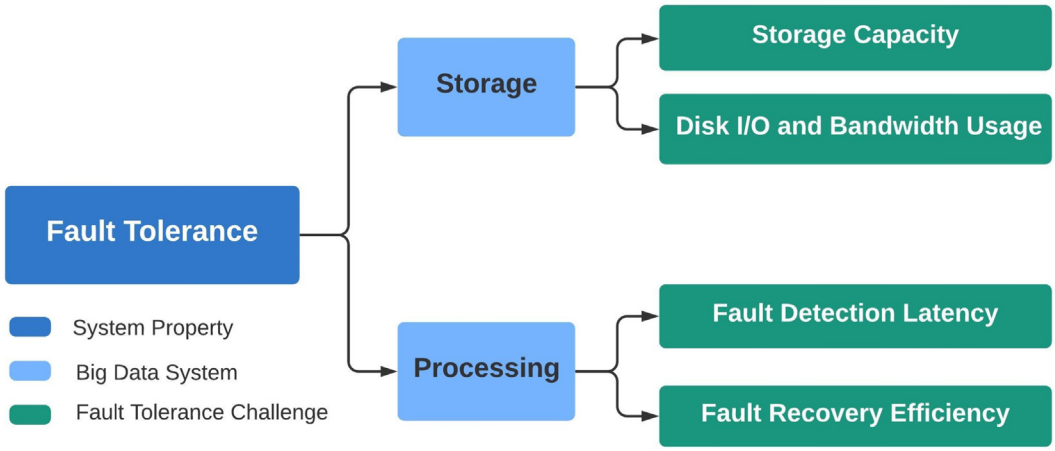


Fig. 3. Classification of fault tolerance challenges in big data systems.

replicas while meeting the availability requirement. CDRM uses a mathematical model to capture the relationship between the number of replicas and the data availability requirement. The model determines the minimum replicas by comparing the average failure rate and the existing number of replicas with the expected availability requirements given by a user. When the requirement is unsatisfied, CDRM dynamically creates new replicas. The system has been implemented and integrated with HDFS. The experimental results showed that the adaption of this approach maintains an optimal number of replicas with a stable storage. Another solution; namely, the cost-effective dynamic incremental replication (CIR), was proposed by Li et al. [56]. CIR dynamically determines the number of replicas on the basis of a reliability model proposed by the same study. The reliability model solves reliability functions that provide minimal replica estimations. These functions estimate the number of replicas needed on the basis of the reliability parameters, such as the probability of fault and storage duration. The functions then demonstrate whether the current number of replicas assures the requirement of data reliability. Thus, new replicas are created incrementally when the replica number does not satisfy the reliability requirement. The study also demonstrated that CIR substantially reduces data storage consumption when the data are stored for a short time. However, CIR is only based on the reliability parameters and the pricing model of Amazon S3; thus, it is unsuitable for Google clusters with a much higher fault rate than the Amazon S3 storage. Similarly, Li and Yang [19] attempted to minimise the number of replicas while meeting the data reliability requirement. The authors presented a cost-effective reliability management mechanism (PRCR) based on a generalised mathematical model that calculates the data reliability with variable disk failures. PRCR applies an innovative proactive replica checking approach to ensure data reliability whilst maintaining data with a minimum number of replicas. The evaluation results have demonstrated that PRCR can manage a large amount of data and significantly reduce storage space consumption at negligible overhead.

Multi-objective strategies were proposed in [58,59] to overcome the storage capacity overhead. In both the studies, the authors argued that factors such as latency, number of replicas, and storage cost were conflicting with one another in the storage system. For example, achieving better latency and data availability require creating additional replicas, and this requirement invariably increases the storage capacity and the cost. Similarly, placing the replicas on the most stable nodes may not be ideal from a latency minimisation perspective. Therefore, multi-objective optimisation deals with these conflicting objectives were solved by evolving a set of solutions that compromised these conflicting objectives. Hassan et al. [59] proposed two algorithms; namely, the multi-objective evolutionary (MOE) and the multi-objective randomised greedy (MORG), for deciding the number of replicas and their placement in a storage system. The MOE algorithm can provide high-quality solutions albeit a longer execution time. By contrast, the MORG algorithm is characterised by its superior computational efficiency and yields solutions that are of comparable quality. Both algorithms aim for a trade-off among three objectives; namely, latency, reliability, and storage consumption, to achieve the optimum number of replicas and efficient replica placement. Following this work, Long et al. [58] also proposed a multi-objective solution called the multi-objective optimised replication management (MORM) scheme based on the improved artificial immune algorithm [61]. The MORM depends on tracking the historical system information and feeding it to a storage engine to maintain and optimise five main objectives: file availability, mean service time, load variance, energy consumption, and latency. Long et al. [58] formulated each respective objective in a way that yields close to the optimal values towards finding the ideal replication factor and the replication layout in the storage system. The MORM was evaluated and compared with the default replication strategy of HDFS. The results conclusively demonstrated that the MORM is better in terms of storage consumption and performance.

Works by Huang et al. [52,60] focus on optimising the erasure coding in the storage system to achieve a minimal disk I/O and the network traffic consumption when reconstructing lost data. Huang et al. [60] proposed a basic pyramid coding (BPC) and used erasure codes to reduce the reconstruction read costs in the storage system. BPC aims to improve the read performance because it greatly affects the overhead storage system performance. The read performance is a primary concern in most storage systems because they normally suffer more read operations than writes. The authors reported that BPC can be formed from (11, 8) MDS. For instance, the 8 data blocks of the MDS can be divided into two identical size units. In this case, the number of the parity data is 3 according to k – n. Accordingly, the global parities can be considered 2 out of the 3, and they can be kept unchanged. Local parities reconstruct two new redundant blocks from the two separately divided units. The experimental results of this study showed that the proposed solution can significantly decrease the read cost by 50% because the local parities decrease the disk I/O when reconstructing the lost data. Following this work, Huang et al. [52] introduced a local reconstruction coding (LRC). In this work, the authors aimed to

reduce the number of erasure coding fragments when performing read operations for reconstructing offline data fragments while maintaining low storage overhead in WAS. LRC is represented as (k, l, r), where k is the divided data fragments by LRC into l local groups. This approach encodes local parities one for each local group l and r global parities. The local group can decode any signal lost data fragment from k/l fragments. LRC can decrease the bandwidth and I/O traffic when reconstructing offline data fragments because the latency for decoding fragments is improved. In the Facebook storage cluster, RSC is used as a standard fault-tolerance approach to avoid the storage overhead caused by three-replica-based storage systems. However, the high cost of RSC is often considered the unavoidable price for storage efficiency and reliability. Sathiamoorthy et al. [43] proposed an efficiently repairable ensure code solution called XORing the Elephants. This approach provides a higher efficiency in terms of disk I/O, network bandwidth, and reliability than the RSC. The proposed solution was implemented on a modified version of HDFS for the Facebook storage clusters. This solution incorporates a local parity that was defined as the LRC (10, 6, 5). Every 5 data blocks will be defined as a local parity to tolerate any lost block by reconstructing it with the remaining blocks in the group locally. The experimental results of this study showed that this solution can significantly improve the disk I/O and network bandwidth performance compared with the RSC when reconstructing the parity with 14% storage overhead. In this case, this overhead is further considered responsible for their scenarios.

4.2. Challenges and proposed solutions for the processing system

During data processing, the system experiences various permanent and transient faults due to numerous errors from the CPU, network, memory, application code, or timeout as well as an unexpected response. These are common faults specifically in the data processing environments where the data sources and processing are distributed among different servers. Thus, the big data processing systems demand a robust fault tolerance strategy that enables them to reactively detect and recover faults at runtime before having a complete failure of the service.

4.2.1. Fault detection latency

The current implementations of most big data systems are based on the heartbeat detection approach [29,62,63]. During the heartbeat detection process, high detection latency happens when 1) delay occurs on the static timeout value of the heartbeats, which message is received after the entire system has been affected, and 2) the only indication of possible permanent faults provided when partially or transient faults may still be able to reset the timer; thus, it leads to a significant performance overhead.

4.2.2. Fault recovery efficiency

A fault of a single task negatively affects all the healthy running data processing tasks and leads to an unpredictable execution time. The current fault recovery approaches based on retry mechanisms, such as speculation execution that involves a high computation and suffers from various limitations. The high computation occurs when the scheduler of the resource manager takes a long time to launch the recovery tasks due to an overloaded or full capacity of associated resources. Such a delay leads to the misplacing or re- executing the faulty tasks from scratch.

4.2.3. Proposed solutions

Works by Gupta et al [64], Pinto et al. [65], Rosa et al. [66], and Soualhia et al. [24,67] proposed fault prediction approaches to predict the faults earlier rather than relying on the default explicit heartbeat approach used in big data systems. On the other hand, works by Yildiz et al. [68], Kadirvel et al. [69], Fu et al. [18], Memishi et al. [70], Liu and Wei [71], Zhu et al. [72], and Yuan et al. [73] proposed fault recovery approaches to provide efficient recovery. The authors of [18,68–70] intended to address the challenge from the resource manager perspective by adopting fault tolerance approaches to the resource scheduler. Further, the authors of [71–73] attempted to apply checkpointing approaches to provide an efficient fault recovery by checkpointing the state of the faulty tasks.

Gupta et al. [64] proposed a predictive and feedback-based system called Astro. The system has a predictive model that collects information about the cluster behaviour, including 163 metrics, such as CPU user per second, byte read per second, and thread count. The predictive model performs a principal component analysis on each node of the cluster to detect possible future faults and reduce the number of predicted variables of the collected metrics. Astro posts the possible faults to its built-in scheduler via APIs. The scheduler constructs suitable decisions to reject or allocate additional resources for improving cluster reliability and throughput. The proposed system has been integrated with a Hadoop cluster. The experimental results showed that Astro improves the resource utilisation by 64.23% and the execution time by 26.68% under the faults compared to the traditional fault detection approach used by Hadoop. Next, Pinto et al. [65] reported that the current heartbeat approach provides a late fault alert after the entire cluster is affected. This condition increases the job execution time and decreases the overall cluster throughput. Therefore, the study proposed a fault detection solution based on performance monitoring and fault prediction. Firstly, the proposed solution collects statistical data of the cluster resources on the basis of a popular performance monitoring framework called Ganglia [74]. Secondly, the collected data are used to train a support vector machine (SVM) model for predicting the upcoming faults before they occur. The SVM model then declares the suspected faults in every available node. Accordingly, the client has an option to terminate or ignore the suspected faults in a node. The simulation results of this study showed that the proposed solution improves the fault detection performance by predicting the faults before they occur; thus, it improves the job execution time under faults. Similarly, Rosa et al. [66] proposed a two-level neural network (NN) model and resource conservation policy for failure prediction and recovery. The prediction model predicts jobs and task-dependent failure patterns. The prediction results contain the predicted tasks that will fail. In the second stage, the resource conservation policy proactively terminates the given tasks to reduce resource consumption. The evaluation results of this study showed that this solution reduces 49%, 45%, 18%, and 26% of the CPU, RAM, disk consumptions, and computational time, respectively, while incorrectly terminating only 1% of successful tasks. Soualhia et al. [67] proposed an adaptive failure-aware scheduler (ATLAS). ATLAS predicts task faults and adjusts the scheduling decisions dynamically to reduce task fault occurrences and improve performance. The methodology of ATLAS contains three main steps that provide failure-aware scheduling decisions. Firstly, job and task attributes are extracted by running different workloads on several nodes. Secondly, the correlation between job and task scheduling results is identified by analysing their dependencies. Thirdly, statistical predictive machine learning based on the identified correlations is used to predict the possible task faults. Numerous regression and classification algorithms, such as the general linear model, random forest, NN, boost, tree, and conditional tree, are applied to validate the predictive model measurement in terms of accuracy, precision, recall and error. The study results showed that ATLAS minimises the failure rate of jobs and tasks and reduces the execution time and resource utilisation. However, ATLAS may provide incorrect predictions that violate the scheduler decisions if unfore-

seen types of faults occur. Soualhia et al. [24] conducted further research and proposed a new version of ATLAS called ATLAS+. In ATLAS+, the study focused on providing efficient and fault tolerant scheduling policies that prevent the performance overhead under the dynamic behaviour of the cluster environment. ATLAS+ consists of three main components: task failure prediction, dynamic failure detection and scheduling policy. The task failure prediction collects historical logs about the executed jobs with their tasks on the cluster environment and applies machine learning algorithm to predict possible failures. The authors conducted an experimental evaluation to compare different machine learning algorithms, including Tree, Boost, Glm, CTree, random forest, and NN, for determining the most suitable for the MapReduce framework in terms of accuracy, precision, recall, error percentage and execution time. The results showed that the random forest algorithm is fit for predicting the Map and Reduce task failures. Another issue realized by the same study is that the static timeout value of the heartbeat detection approach leads the scheduler to assign tasks to a faulty node if the node fails before the timeout value expires. Therefore, a dynamic failure detection approach was proposed by Soualhia et al. [24]. This approach dynamically adjusts the timeout value of the heartbeat to allow low latency in fault detection and avoid wasting time and resources when assigning tasks to a faulty node. Thirdly, another scheduling policy was adopted in ATLAS+, and it enables the scheduler to make an optimal strategy for reducing or recovering faults. ATLAS+ has been integrated with Hadoop MapReduce. The experimental results showed that ATLAS+ reduces the number of failed jobs by up to 43% and the number of failed tasks by up to 59%. ATLAS+ also reduces the total execution time of jobs and tasks and the CPU and memory usage compared with the default scheduler of Hadoop.

Yildiz et al. [68] proposed a failure-aware scheduling strategy called Chronos, which enables an early and smart action for quick fault recovery. Chronos involves a lightweight pre-emption technique for allocating resources for the recovery task. The resource allocation collects a list of failed tasks along with their nodes that host the task data and sorts them according to job priority. This strategy then compares the priorities of failed tasks with recovery tasks to perform task pre-emption actions, such as kill or suspend. The experimental results of this study showed that Chronos performs correct scheduling behaviour under faults in a few seconds. However, this strategy may lead to resource wastage and performance overhead if it performs incorrect pre-emption actions. Next, Kadirvel et al. [69] proposed a technique that provides early fault detection and management for MapReduce called fault-managed Map-Reduce (FMR). FMR enables the detection and reduction of faults that extend the job execution time and lead to inefficient resource consumption. FMR adopts a prediction model based on sparse coding to predict the computation time of running tasks on each available node. The authors claimed that the required time for training the prediction model is only a few seconds. FMR uses a closed-loop for the fault management that provides dynamic resource scaling. FMR uses the prediction model results to estimate the MapReduce job execution time and compares the costs of execution time and additional resources required for removing failures. The comparison results showed that this technique makes an appropriate scaling decision to enable minimal computing overhead during failures. However, the disadvantage of this technique is the necessity to use decentralised and local training of the models on every available node. This requirement may introduce inefficient resource usage while training these models. Fu et al. [18] proposed a framework that contains a central fault analyser (CFA), which is a new speculation mechanism and scheduling procedure. The CFA monitors the failures at runtime and provided failure analysis to FARMS for a speculation decision. In the case of a failed node, FARMS provides all the affected tasks, including the completed ones, to launch SE tasks. After completing the SE, the Reduce tasks are notified to fetch the output from the new tasks rather than the primary tasks executed on the failed node. FARMS speculates all the affected tasks at once when all the tasks are running on a struggler node. The proposed scheduling procedure is called fast analytics scheduling. This procedure provides a trade-off between fault detection performance and resource consumption based on a dynamic threshold. The experimental results of this study showed that this framework improves the performance of the existing YARN SE when handling node failures. The authors demonstrated that the proposed framework is specifically designed for small jobs with a short turnaround time. Furthermore, Memishi et al. [70] observed that the efficiency of fault recovery is dependent on the latency of fault detection. Thus, the authors attempted to address the challenge from the detection and recovery perspectives. The study proposed three abstractions; namely, high relax failure detector (HR-FD), medium relax failure detector (MR-FD) and low relax failure detector (LR-FD). HR-FD estimates the completion time of the workload and then adjusts the timeout value of the heartbeat messages to decrease the latency of detecting faults [70]. However, the timeout value remains static; thus, MR-FD was proposed. MR-FD calculates the progress score of each workload to dynamically adjust the timeout value. LR-FD was proposed to expend MR-FD by monitoring the timeout of tasks and workers. MR-FD launches a speculative task on another healthy node when it detects un unexpected behaviour caused by a task or a worker. The simulation results of this study showed that the proposed abstractions outperform the existing heartbeat approach of Hadoop in terms of performance. However, the proposed abstractions in this study were only evaluated on a simulated environment. Furthermore, Liu and Wei [71] proposed a checkpoint-based active replication method to improve the current fault tolerance strategy of Hadoop in terms of resource usage and the job execution time during data processing under faults. The proposed solution consists of two checkpoint files created before the execution of Map tasks. The first checkpoint file is a local file responsible for checkpointing the output of each running Map task. The second checkpoint file is called the global index file, which is responsible for reconstructing the intermediate results across nodes to reduce the re-execution time in case of node failures. The global index can be stored on HDFS for high availability. The experimental results of this study showed that the proposed solution improves the resource utilisation and overall execution time of Hadoop under failures. Another study by Zhu et al. [72] proposed a novel fault tolerance strategy called fast recovery MapReduce, which uses a combination of distributed checkpointing and proactive push mechanism to provide low-latency when recovering from task and node faults [72]. The checkpointing allows periodical recording of the computational progress for each Map task as a checkpoint. In the case of a fault, the recovered task can continue computing on the basis of the recent checkpoint without the necessity to recompute the entire data block. The checkpoint data of the tasks are stored in a distributed data storage. The distributed data storage can be HDFS or distributed memory. A proactive push mechanism is also proposed in this solution in which the computational results of Map tasks are proactively transferred to the Reducers when they are produced without waiting for the completion of Map progress. This proactive push mechanism is utilised to allow the use of any data produced by the Map tasks before they fail. The evaluation results of the study showed that this method improves the performance by 55% during a task fault and by 45% during a node fault compared with the original Hadoop fault recovery strategy. Yuan et al. [73] proposed a strategy that offers efficient resource utilisation under task faults. This strategy is customised for MapReduce, and it specifically monitors the nodes that launch Map task faults to isolate whether they are faulty or strugglers on the basis of

the node performance. This strategy uses external storage as a cache server to store the output of the intermediate results of Map tasks to be accessible in case of failures. The cache server helps maintain the retrieval of the intermediate results of the complete Map tasks in case the entire node failed to prevent executing the complete tasks from scratch. The simulations of this study showed that this strategy maintains the high performance of the cluster while avoiding the re-execution of complete Map tasks, especially for the large MapReduce jobs. However, the study did not show the influence of the proposed strategy on resource utilisation while detecting Map tasks and transferring the intermediate task results to the cache server.

5. Analysis

Several researchers are increasingly concerned with fault tolerance in big data storage and processing systems for high reliability and efficient performance and resource utilisation under faults. This work reviewed several studies related to fault tolerance in big data storage and processing systems. Table 1 shows an identification of the conducted studies, while Table 2 provides a summary of the reviewed studies with the challenges, solutions, and the experimental setup. According to Table 1, there are more studies that have been recently conducted on fault tolerance in the processing system than in the storage system. The results also showed that the research efforts to improve the fault tolerance in the processing systems are growing. However, the studies for addressing the fault tolerance challenges in the big data storage systems have been inactive in recent years. This phenomenon may be due to the significant cost of the processing system's computational resources. Furthermore, we categorised the proposed solutions according to their fault tolerance approaches as projected in Table 2 and these categories are further analysed and discussed in the following subsections. Tables 3 and 4 provide the advantages and the disadvantages of the proposed solutions.

5.1. Analysis of the proposed solutions for big data storage system

The reviewed studies in Table 2 showed that the proposed fault tolerance solutions can be classified into different solution categories to address the challenges of big data storage and processing systems. The main three solution categories to address the storage system challenges are reliability trade-off, multi-objective optimisation and erasure coding optimisation. Reliability trade-off solutions were proposed to investigate the relationship between the storage system data reliability and the proper number of replicas for decreasing the storage capacity overhead. Next, the multi-objective optimisation solutions have focused on optimising various factors, such as data availability, latency, and storage cost, that influence the storage capacity overhead challenge. Next, erasure coding optimisation solutions were proposed with various algorithms that aim to mitigate the challenges of the high disk and I/O usage in the storage system. The advantages and disadvantages of the three solution categories are elaborated in Table 3.

Reliability trade-off solutions [19,56,57] have reduced the number of the replicated data by investigating the relationship between the data reliability requirements and the number of replicas. However, they have not drastically overcome the issue when terabytes or petabytes of data must be replicated for data availability. Furthermore, the studies have not focused on other affected factors, such as data locality and access latency, that would decrease the performance of the entire storage system if the replicas have not been placed properly. By contrast, multi-objective optimisation solutions [58,59] deal with a set of objectives, such as replica number, replica placement, energy consumption, and storage cost, that compromise the storage capacity challenge. They have achieved the optimum number of replicas with the minimum performance overhead compared with the current replication strategies of big data storage systems, such as HDFS. In summary, the proposed solutions based on the replication approach have minimised the number of data redundancy in the storage systems. However, none of them can minimise the storage capacity overhead without sacrificing reliability.

Erasure coding optimisation solutions [43,52,60] aimed to optimise the performance of erasure coding in popular storage systems, such as HDFS and WAS. Huang et al. [52] reduced the code reconstruction read cost by 50%. Sathiamoorthy et al. [43] reduced the disk I/O usage and network traffic during the reconstruction. However, the repair overhead of erasure coding is approximately 10 times higher than the replication. All the studies have only focused on the permanent disk faults where there are transient faults that also lead to data unavailability for a certain period. A hybrid fault tolerance strategy based on replication and erasure coding for the storage system that can provide minimum storage overhead and high disk I/O performance has not yet been achieved.

Table 1
Quick identification of fault tolerance challenges in big data storage and processing systems addressed by previous studies.

Reference	Storage	Processing	SC	DIOBU	FDL	FRE
[24]		✓			✓	
[72]		✓				✓
[70]		✓			✓	✓
[65]		✓			✓	
[68]		✓				✓
[19]	✓		✓			
[67]		✓				✓
[71]		✓				✓
[66]		✓			✓	✓
[58]	✓		✓			
[64]		✓			✓	
[60]	✓			✓		
[43]	✓			✓		
[69]		✓				✓
[73]		✓				✓
[52]	✓			✓		
[56]	✓		✓			
[57]	✓		✓			
[59]	✓		✓			

Note: SC: Storage Capacity, DIOBU: Disk I/O and Bandwidth Usage, FDL: Fault Detection Latency and FRE: Fault Recovery Efficiency.

Table 2
Summary of the challenges, solutions, and experimental setup of previous studies for achieving fault tolerance in big data storage and processing systems.

Fault Tolerance Strategy	Challenge	Fault Tolerance Solution Category	Ref.	Experimental Environment	Fault Type	Evaluation Metrics
Fault Recovery	Storage Capacity	Reliability Trade-off	[57]	HDFS	Permanent	Availability, Failure Rate and Replica Number
			[56]	Amazon S3	Permanent	Reliability and Replica Number
			[19]	Amazon EC2 and S3	Permanent	Reliability and Replica Number
		Multi-objective Optimisation	[59]	Peer-to-peer Overlay	Permanent	Reliability, Storage Capacity and Latency
			[58]	CloudSim	Permanent	File Unavailability, Service Time, Load Variance, Energy Consumption and Latency
	Disk I/O and Bandwidth Usage	Erasure Code Optimisation	[60]	WAS	Permanent	Disk Read Overhead
			[52]	WAS	Permanent	Latency, I/O and Bandwidth
			[43]	Facebook HDFS	Permanent	Byte Read, Network Traffic and Repair Time
Fault Detection	Fault Detection Latency	Fault Prediction	[64]	Hadoop Cluster	Permanent	Compute Resource Usage and Execution Time
			[65]	Hadoop and Ganglia	Transient	CPU Usage
			[66]	Google Cluster Trace	Permanent	CPU, RAM, Disk Usage and Execution Time
			[67]	Amazon EMR	Transient and Permanent	CPU, RAM, HDFS I/O, Number of Failed Jobs and Tasks and Execution Time
			[24]	Amazon EMR	Transient and Permanent	CPU, RAM, HDFS I/O, Number of Failed Jobs and Tasks and Execution Time
Fault Recovery	Fault Recovery Efficacy	Preemption	[68]	Hadoop on Grid'5000	Permanent	Data Locality and Execution Time
		Dynamic Resource Scaling	[69]	Hadoop on IBM Blade Servers	Transient and Permanent	Execution Time
		Speculative Execution	[18]	Hadoop YARN	Transient	Execution Time
		Speculative Execution	[70]	Statistical-based Simulation	Transient	Execution Time
Fault Recovery	Fault Recovery Efficacy	Checkpointing	[71]	Hadoop Cluster	Permanent	Storage Capacity, Bandwidth and Execution Time
			[72]	Hadoop Cluster	Permanent	Execution Time
			[73]	CloudSim	Permanent	Execution Time

Table 3
Advantages and disadvantages of the proposed fault tolerance solutions for big data storage systems.

Solution Category	Advantages	Disadvantages
Reliability Trade-off	It reduces storage cost by minimising the number of replicas based on the reliability requirement.	Specific system parameters must be set to trade-off between storage usage and reliability.
Multi-Objective Optimisation	It reduces storage cost by considering multiple objectives, such as efficient replica placement for the minimum latency and minimising the number of the replica with fair reliability.	It is unsuitable for the storage system that requires dynamic changes in its infrastructure.
Erasure Coding Optimisation	It offers efficient data redundancy with a high level of fault tolerance.	It involves high disk I/O usage and network bandwidth overhead especially if the level of data distribution is high.

5.2. Analysis of the proposed solutions for big data processing system

The proposed solutions of the conducted studies to address fault tolerance challenges in the processing system are categorised into fault prediction, preemption, dynamic resource scaling, speculative execution, and checkpointing. The advantages and disadvantages of these solution categories are projected in Table 4.

Fault prediction solutions aim to address the lack of fault detection in the processing systems by providing early fault prediction before the fault occurs. Proposed solutions in [24,64–67] have reduced the number of faults and prevented an expected performance violation by suspecting faults before they occur. These solutions are mainly focused on permanent faults because these faults lead to high-performance overhead and resource consumption. Most fault prediction solutions are evaluated in real big data processing environments, including Hadoop cluster or Amazon EMR. The evaluation metrics used to measure the performance in the fault prediction solutions are the workload execution time and compute resource consumption. Although fault prediction solutions offer early fault detection and improve the system

Table 4
Advantages and disadvantages of the proposed fault tolerance solutions for big data processing systems.

Solution Category	Advantages	Disadvantages
Fault Prediction	It solves the challenge of high detection latency by alerting a fault before it occurs.	It involves a predicative model that needs to be fed with various system metrics and previously generated fault.
Preemption	It adjusts the task scheduler decision by offering efficient fault recovery in terms of computation for high priority tasks.	It requires a task pre-emption policy that considers task correlations, priority, execution time, and resource utilisation.
Dynamic Resource Scaling	It allows the system to dynamically scale up for providing a healthy environment for re-executing a faulty task.	The computation time of the running tasks on each active node must be calculated to provide a suitable resource allocation.
Speculative Execution	It ensures a high level of fault tolerance with a fair performance by duplicating the running tasks.	It requires additional computational resources.
Checkpointing	It provides a high level of fault tolerance for real-time data processing by recording the state of each active node or task and replicating it on another stable environment.	The consistency of the replicated state is challenging, and the cost of CPU consumption is high.

performance and resource consumption compared with the default fault detection approach, they still cannot predict every possible type of fault, especially in large-scale environments where faults are unforeseeable.

Furthermore, the efficiency of fault recovery in the processing systems is significantly challenging in terms of performance because such systems execute their workloads on the distributed servers in parallel. One faulty task during the processing time affects the overall performance of the workload if it has been recovered improperly. Thus, solutions in the three categories including speculative execution, preemption, and dynamic resource scaling reported that the efficiency of fault recovery is mostly related to the task scheduler of the resource manager [18,68–70]. Specifically, if a fault occurs when the cluster is running at full computational capacity, then the scheduler must wait for an empty slot to re-execute the faulty task on another healthy node. In this way, it extends the execution time of the entire workload, which degrades the overall performance and resource consumption. Although the proposed solutions have improved the efficiency of the existing fault recovery approaches in the respective systems, they have only focused on the gab from one individual view, which is fault recovery. Nevertheless, fault recovery depends on the result of fault detection. In other words, if a delay in detecting a fault occurs, then the fault recovery action would be negatively affected because the recovery decision relies on the availability of the computational resources that are dynamically changing at the processing time; by contrast, fast fault detection leads to a fast and efficient fault recovery. Only Memishi et al. [70] and Rosa et al. [66] focused on fault detection and recovery challenges in their fault tolerance solutions. Rosa et al. [66] applied a fault prediction solution to predict faults and upon the prediction result, provided a preemption approach for terminating the suspected faults before they exist for recovery. Likewise, Memishi et al. [70] proposed a solution that adjusts the timeout value of the heartbeat messages to accelerate the speed of detecting fault; thus, this method avoids any unexpected behaviour of the speculative execution in the fault recovery stage. Moreover, the solutions under the checkpointing category also aim to address the challenge of fault recovery efficiency [71–73]. These solutions allow periodical recording of the computational progress of each executed task to avoid inefficiently re-executing the faulty task from scratch. This approach offers huge savings in terms of execution time and computational consumption. For instance, Zhu et al. [72] showed that the checkpointing approach improves the overall performance by 55% during task faults and by 45% during node faults. However, the consistency of checkpointing the state of the running tasks is challenging. Although all the proposed solutions can be applied in the common big data processing systems to mitigate the highlighted challenges, most of the studies are only focused on the fault tolerance challenges observed in MapReduce and YARN systems.

6. Future directions

This review has shown the growing demand to implement fault tolerance in big data systems. In the storage systems, various challenges, such as the mechanisms by which to minimise the storage overhead and disk read and write costs and reduce the network bandwidth, have been introduced when offering fault tolerance. The following challenges in the processing systems have also been introduced when providing fault detection and fault recovery: quick and precise detection of faults before causing damage or performance penalties and recovery of the system into its normal state within a minimum performance overhead and optimum resource usage. The following directions are listed to address the above-mentioned challenges in the future research.

- A combination of fault prediction and replication approach for the storage systems' fault tolerance is needed. This combination can use fault prediction to predict faults. Upon prediction, replication can be performed dynamically to create the minimum number of replicas without sacrificing data reliability, and availability.
- A hybrid technique that involves erasure coding and replication can also be applied in the storage systems. Erasure coding is storage efficient compared to replication, and replication provides a saving network traffic and disk I/O. Thus, a hybrid technique that can merge the best features of each fault-tolerance approach is a promising future research.
- An adaptive speculative execution and dynamic heartbeat-based approach for fault detection and recovery that accurately estimates the workload processing time and sets the timeout value on the fly is needed to improve the performance of fault detection in detecting permanent faults. Although heartbeat detection has limitations in detecting transient faults, speculative execution can be adapted to duplicate strugglers by monitoring their processing progress and terminate the slow running tasks dynamically for avoiding extensive computational resource usage.
- A new decentralised method for fault detection and recovery can overcome the fault tolerance challenges in the processing systems. The processing systems determine the locations of data chunks and their backup copies in the storage system. Thus, rather than relying on the cluster's master node in handling fault detection and rescheduling faulty tasks, each node that hosts the redundant data can act as a partner node and establish a peer-to-peer communication with the other node that runs the data processing tasks on the original chunk of data and periodically exchanges heartbeats for fault detection. In the case of a faulty task or node, the partner node quickly takes over and re-executes the affected tasks. With this method, the load of fault detection will be distributed among nodes rather than handled by the master node. The latency of detecting and recov-

ering faults is also expected to be improved. This method can support data locality and can thus reduce network traffic if the redundant data are stored in the same rack that hosts the original data.

7. Limitation

The sources used in this review paper are selected from reliable database engines that include IEEE Xplore, ScienceDirect, Web of Science, and Google Scholar to cover broad aspects of the fault tolerance topic in large-scale systems, specifically big data systems. In addition, this review has covered numerous research papers from journals, conferences, technical reports, and workshops. However, the identification of the fault tolerance challenges is limited to certain factors, such as the time frame of conducting the review and the increased research progress in this area. The research scope is set to two main criteria: the fault tolerance challenges in the storage and the processing systems. These two types of big data systems are commonly used together in big data clusters to enable reliable and fault-tolerant big data solutions. Therefore, scientific articles that have provided fault tolerance solutions for big data systems are only included for the identification of their fault tolerance challenges and reviewing of their solutions. Works that have only focused on fault tolerance or big data system as the main topic are excluded in this review.

8. Conclusion

Fault tolerance plays a predominant role in enabling big data systems to provide reliable data storage and processing services. Big data systems reside in large-scale environments that are exposed to numerous faults. Accordingly, various fault tolerance approaches have been used in big data systems to improve reliability and maintain consistent data processing under faults. Fault tolerance in storage systems relies on data redundancy approaches, including replication and erasure coding. In the meantime, fault tolerance in the processing system requires detecting and recovering faults at the processing time. This paper briefly explained big data systems including the storage and processing layers that are vulnerably subjected to faults and failures. Moreover, this study provided an overview of fault tolerance concepts, including common fault types that occur in large-scale systems and the fault tolerance approaches used to tolerate the faults. Furthermore, this paper classified the common fault tolerance challenges faced by previous studies that proposed fault-tolerance solutions for big data systems. The classified challenges are storage capacity and disk I/O and bandwidth usage in the storage systems and fault detection latency and fault recovery efficiency in the processing systems. For each of the challenges found in big data storage and processing systems, the approaches taken by research are further discussed and analysed. The future research directions are listed, and they can be applied to big data systems or other large-scale systems to improve the efficiency of fault tolerance and overcome the highlighted challenges. The findings derived from this review can serve as a useful guide to facilitate the progress of conducting future research in fault tolerance for big data systems.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We gratefully thank the Malaysia Ministry of Education for the Fundamental Research Grant Scheme provided to us numbered GPF097C-2020.

References

[1] Medhat D, Yousef AH, Salama C. Cost-aware load balancing for multilingual record linkage using MapReduce. Ain Shams Eng J 2020;11:419–33. doi: https://doi.org/10.1016/j.asej.2019.08.009.
[2] Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. J Syst Softw 2017;125:133–51. doi: https://doi.org/10.1016/j.jss.2016.11.037.
[3] Kasu P, Kim T, Um JH, Park K, Atchley S, Kim Y. FTLADS: Object-Logging Based Fault-Tolerant Big Data Transfer System Using Layout Aware Data Scheduling. IEEE Access 2019;7:37448–62. doi: https://doi.org/10.1109/ACCESS.2019.2905158.
[4] MySQL n.d. https://www.mysql.com/ (accessed January 14, 2020).
[5] Dean J, Fellow G. Designs, Lessons and Advice from Building Large Distributed Systems. 2009.
[6] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Commun ACM 2008;51:107–13. doi: https://doi.org/10.1145/1327452.1327492.
[7] Apache Hadoop n.d. https://hadoop.apache.org/ (accessed March 10, 2020).
[8] Jhawar R, Piuri V. Fault Tolerance and Resilience in Cloud Computing Environments. In: Comput. Inf. Secur. Handb.. Elsevier; 2017. p. 165–81. doi: https://doi.org/10.1016/b978-0-12-803843-7.00009-0.
[9] Sharma Y, Javadi B, Si W, Sun D. Reliability and energy efficiency in cloud computing systems: Survey and taxonomy. J Netw Comput Appl 2016;74:66–85. doi: https://doi.org/10.1016/j.jnca.2016.08.010.
[10] Torres-Huitzil C, Girau B. Fault and Error Tolerance in Neural Networks: A Review. IEEE Access 2017;5:17322–41. doi: https://doi.org/10.1109/ACCESS.2017.2742698.
[11] Nabi M, Toeroe M, Khendek F. Availability in the cloud: State of the art. J Netw Comput Appl 2016;60:54–67. doi: https://doi.org/10.1016/j.jnca.2015.11.014.
[12] Fadika Z, Govindaraju M. LEMO-MR: Low overhead and elastic MapReduce implementation optimized for memory and CPU-intensive applications. In: Proc. - 2nd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2010, 2010, p. 1–8. https://doi.org/10.1109/CloudCom.2010.45.
[13] Jin H, Yang X, Sun XH, Raicu I. ADAPT: Availability-aware mapreduce data placement for non-dedicated distributed computing. In: Proc. - Int. Conf. Distrib. Comput. Syst. IEEE; 2012. p. 516–25. doi: https://doi.org/10.1109/ICDCS.2012.48.
[14] Lin H, Ma X, Archuleta J, Feng WC, Gardner M, Zhang Z. MOON: MapReduce on opportunistic eNvironments. In: HPDC 2010 - Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput., 2010, p. 95–106. https://doi.org/10.1145/1851476.1851489.
[15] Costa P, Pasin M, Bessani AN, Correia M. Byzantine fault-tolerant MapReduce: Faults are not just crashes. Proc. - 2011 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2011, 2011, p. 32–9. https://doi.org/10.1109/CloudCom.2011.15.
[16] Ko SY, Hoque I, Cho B, Gupta I. Making cloud intermediate data fault-tolerant. In: Proc. 1st ACM Symp Cloud Comput. SoCC '10. p. 181–92. doi: https://doi.org/10.1145/1807128.1807160.
[17] Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In: EuroSys'10 – Proc. EuroSys 2010 Conf.. p. 265–78. doi: https://doi.org/10.1145/1755913.1755940.
[18] Fu H, Chen H, Zhu Y, Yu W. FARMS: Efficient mapreduce speculation for failure recovery in short jobs. Parallel Comput 2017;61:68–82. doi: https://doi.org/10.1016/j.parco.2016.10.004.
[19] Li W, Yang Y, Yuan D. Ensuring Cloud Data Reliability with Minimum Replication by Proactive Replica Checking. IEEE Trans Comput 2016;65:1494–506. doi: https://doi.org/10.1109/TC.2015.2451644.
[20] Quiane-Ruiz J-A, Pinkel C, Schad J, Dittrich J. RAFTing MapReduce: Fast recovery on the RAFT. In: 2011 IEEE 27th Int. Conf. Data Eng., IEEE; 2011, p. 589–600. https://doi.org/10.1109/ICDE.2011.5767877.
[21] Sangroya A, Bouchenak S, Serrano D. Experience with benchmarking dependability and performance of MapReduce systems. Perform Eval 2016;101:1–19. doi: https://doi.org/10.1016/j.peva.2016.04.001.
[22] Liu J, Shen H. A Low-Cost Multi-failure Resilient Replication Scheme for High Data Availability in Cloud Storage. In: Proc. - 23rd IEEE Int. Conf. High Perform. Comput. HiPC 2016, IEEE; 2017, p. 242–51. https://doi.org/10.1109/HiPC.2016.036.
[23] Bi S, Zhang R, Ding Z, Cui S. Wireless communications in the era of big data. IEEE Commun Mag 2015;53:190–9. doi: https://doi.org/10.1109/MCOM.2015.7295483.
[24] Soualhia M, Khomh F, Tahar S. A Dynamic and Failure-aware Task Scheduling Framework for Hadoop. IEEE Trans Cloud Comput 2018. doi: https://doi.org/10.1109/TCC.2018.2805812. 1–1.
[25] Memishi B, Ibrahim S, Pérez MS, Antoniu G. Fault Tolerance in MapReduce: A Survey, Springer, Cham; 2016, p. 205–40. https://doi.org/10.1007/978-3-319-44881-7_11.

[26] Avci Salma C, Tekinerdogan B, Athanasiadis IN. Domain-Driven Design of Big Data Systems Based on a Reference Architecture. In: Softw. Archit. Big Data Cloud. Elsevier; 2017. p. 49–68. doi: https://doi.org/10.1016/b978-0-12-805467-3.00004-1.
[27] Oussous A, Benjelloun F-Z, Ait Lahcen A, Belfkih S. Big Data technologies: A survey. J King Saud Univ - Comput Inf Sci 2018;30:431–48. doi: https://doi.org/10.1016/J.JKSUCI.2017.06.001.
[28] Ghemawat S, Gobioff H, Leung ST. The google file system. Oper Syst Rev 2003;37:29–43. doi: https://doi.org/10.1145/1165389.945450.
[29] Shvachko K, Kuang H, Radia S, Chansler R. HDFS - The Hadoop distributed file system. In: 2010 IEEE 26th Symp Mass Storage Syst Technol MSST2010. p. 1–10.
[30] Swift - OpenStack n.d. https://wiki.openstack.org/wiki/Swift (accessed March 10, 2020).
[31] Nachiappan R, Javadi B, Calheiros RN, Matawie KM. Cloud storage reliability for Big Data applications: A state of the art survey. J Netw Comput Appl 2017;97:35–47. doi: https://doi.org/10.1016/j.jnca.2017.08.011.
[32] O'Reilly J. Network Storage: Tools and Technologies for Storing Your Company's Data. 2016.
[33] Verma C, Pandey R. Comparative analysis of GFS and HDFS: Technology and architectural landscape. In: Proc. - 2018 10th Int. Conf. Comput. Intell. Commun. Networks, CICN 2018, Institute of Electrical and Electronics Engineers Inc.; 2018, p. 54–8. https://doi.org/10.1109/CICN.2018.8864934.
[34] Apache SparkTM - Unified Analytics Engine for Big Data n.d. https://spark.apache.org/ (accessed March 10, 2020).
[35] Apache Storm n.d. https://storm.apache.org/ (accessed March 10, 2020).
[36] Samza n.d. http://samza.apache.org/ (accessed March 10, 2020).
[37] Apache Flink: Stateful Computations over Data Streams n.d. https://flink.apache.org/ (accessed March 10, 2020).
[38] The Apache Software Foundation. Welcome to Apache Flume — Apache Flume. Apache Softw Found 2012. https://flume.apache.org/ (accessed June 19, 2019).
[39] Soualhia M, Khomh F, Tahar S. Task Scheduling in Big Data Platforms: A Systematic Literature Review. J Syst Softw 2017;134:170–89. doi: https://doi.org/10.1016/j.jss.2017.09.001.
[40] Inoubli W, Aridhi S, Mezni H, Maddouri M, Nguifo Mephu E. An experimental survey on big data frameworks. Futur Gener Comput Syst 2018;86:546–64. doi: https://doi.org/10.1016/J.FUTURE.2018.04.032.
[41] Nelson VP. Fault-Tolerant Computing: Fundamental Concepts. Computer (Long Beach Calif) 1990;23:19–25. https://doi.org/10.1109/2.56849.
[42] Al-Kuwaiti M, Kyriakopoulos N, Hussein S. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. IEEE Commun Surv Tutorials 2009;11:106–24. doi: https://doi.org/10.1109/SURV.2009.090208.
[43] Sathiamoorthy M, Asteris M, Papailiopoulos D, Dimakis AG, Vadali R, Chen S, et al. XORing elephants: Novel erasure codes for big data. Proc VLDB Endow 2013;6:325–36. doi: https://doi.org/10.14778/2535573.2488339.
[44] Ayari N, Barbaron D, Lefevre L, Primet P. Fault tolerance for highly available internet services: concepts, approaches, and issues. IEEE Commun Surv Tutorials 2008;10:34–46. doi: https://doi.org/10.1109/COMST.2008.4564478.
[45] Wang H, Chen H, Du Z, Hu F. BeTL: MapReduce Checkpoint Tactics Beneath the Task Level. IEEE Trans Serv Comput 2016;9:84–95. doi: https://doi.org/10.1109/TSC.2015.2453973.
[46] Zaharia M, Konwinski A, Joseph AD, Katz R, Stoica I. Improving MapReduce performance in heterogeneous environments. In: Proc. 8th USENIX Symp. Oper. Syst. Des. Implementation, OSDI 2008, 2019, p. 29–42.
[47] Soualhia M, Khomh F, Tahar S. Predicting scheduling failures in the cloud: A case study with google clusters and hadoop on Amazon EMR. In: Proc - 2015 IEEE 17th Int Conf High Perform Comput Commun 2015 IEEE 7th Int Symp Cybersp Saf Secur 2015 IEEE 12th Int Conf Embed Softw Syst H 2015, p. 58–65. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.170.
[48] Kalia K, Gupta N. Analysis of hadoop MapReduce scheduling in heterogeneous environment. Ain Shams Eng J 2020. doi: https://doi.org/10.1016/j.asej.2020.06.009.
[49] Ousterhout J, Gopalan A, Gupta A, Kejriwal A, Lee C, Montazeri B, et al. The RAMCloud storage system. ACM Trans Comput Syst 2015;33:1–55. doi: https://doi.org/10.1145/2806887.
[50] Data redundancy in Azure Storage | Microsoft Docs. https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy (accessed March 10, 2020).
[51] Apache. Apache Hadoop 3.0.0-alpha2 – HDFS Erasure Coding 2017. https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html (accessed May 6, 2020).
[52] Huang C, Simitci H, Xu Y, Ogus A, Calder B, Gopalan P, et al. Erasure coding in windows Azure storage. In: Proc. 2012 USENIX Annu. Tech. Conf. USENIX ATC 2012, 2019, p. 15–26.
[53] Carbone P, Ewen S, Haridi S, Katsifodimos A, Markl V, Tzoumas K. Apache Flink: Unified Stream and Batch Processing in a Single Engine. Data Eng 2015;36:28–38. doi: https://doi.org/10.1109/IC2EW.2016.56.
[54] Xu F, Wang Y, Ma X. Incremental encoding for erasure-coded cross-datacenters cloud storage. Futur Gener Comput Syst 2018;87:527–37. doi: https://doi.org/10.1016/j.future.2018.04.047.
[55] Cook J, Primmer R, de Kwant A. Comparing cost and performance of replication and erasure coding. Hitachi Rev 2013;63.
[56] Li W, Yang Y, Yuan D. A novel cost-effective dynamic data replication strategy for reliability in cloud data Centres. In: Proc. - IEEE 9th Int. Conf. Dependable, Auton. Secur. Comput. DASC 2011. p. 496–502. doi: https://doi.org/10.1109/DASC.2011.95.
[57] Wei Q, Veeravalli B, Gong B, Zeng L, Feng DCDRM. A cost-effective dynamic replication management scheme for cloud storage cluster. In: Proc. - IEEE Int. Conf. Clust. Comput. ICCC. p. 188–96. doi: https://doi.org/10.1109/CLUSTER.2010.24.
[58] Long SQ, Zhao YL, Chen W. MORM: A Multi-objective Optimized Replication Management strategy for cloud storage cluster. J Syst Archit 2014;60:234–44. doi: https://doi.org/10.1016/j.sysarc.2013.11.012.
[59] Al-Haj Hassan O, Ramaswamy L, Miller J, Rasheed K, Canfield ER. Replication in overlay networks: A multi-objective optimization approach. Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng., vol. 10 LNICST, 2009, p. 512–28. https://doi.org/10.1007/978-3-642-03354-4_39.
[60] Huang C, Chen M, Li J. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. ACM Trans Storage 2013;9:1–28. doi: https://doi.org/10.1145/2435204.2435207.
[61] Zhang J. Artificial immune algorithm to function optimization problems. In: 2011 IEEE 3rd Int. Conf. Commun. Softw. Networks, ICCSN 2011. p. 667–70. doi: https://doi.org/10.1109/ICCSN.2011.6014177.
[62] Douglas C, Lowe J, Malley OO, Reed B. Apache Hadoop YARN : Yet Another Resource Negotiator n.d.
[63] Toshniwal A, Donham J, Bhagat N, Mittal S, Ryaboy D, Taneja S, et al. Storm@twitter. In: Proc 2014 ACM SIGMOD Int Conf Manag Data - SIGMOD '14 2014, p. 147–56. https://doi.org/10.1145/2588555.2595641.
[64] Gupta C, Bansal M, Chuang T-C, Sinha R, Ben-romdhane S. Astro: A predictive model for anomaly detection and feedback-based scheduling on Hadoop. 2014 IEEE Int. Conf. Big Data (Big Data), IEEE; 2014, p. 854–62. https://doi.org/10.1109/BigData.2014.7004315.
[65] Pinto J, Jain P, Kumar T. Hadoop distributed computing clusters for fault prediction. 2016 Int. Comput. Sci. Eng. Conf., IEEE; 2016, p. 1–6. https://doi.org/10.1109/ICSEC.2016.7859903.
[66] Rosa A, Chen LY, Binder W. Catching failures of failures at big-data clusters: A two-level neural network approach. In: 2015 IEEE 23rd Int Symp. Qual. Serv.,. IEEE; 2015. p. 231–6. doi: https://doi.org/10.1109/IWQoS.2015.7404739.
[67] Soualhia M, Khomh F, Tahar S. ATLAS: An AdapTive faiLure-Aware Scheduler for Hadoop. In: 2015 IEEE 34th Int. Perform. Comput. Commun. Conf.. IEEE; 2015. p. 1–8. doi: https://doi.org/10.1109/PCCC.2015.7410316.
[68] Yildiz O, Ibrahim S, Antoniu G. Enabling fast failure recovery in shared Hadoop clusters: Towards failure-aware scheduling. Futur Gener Comput Syst 2017;74:208–19. doi: https://doi.org/10.1016/j.future.2016.02.015.
[69] Fortes SK and JH and J a B, Kadirvel S, Ho J, Fortes JAB. Fault Management in Map-Reduce Through Early Detection of Anomalous Nodes. In: Proc 10th Int Conf Auton Comput (ICAC 13) 2013, p. 235–45.
[70] Memishi B, Pérez MS, Antoniu G. Failure detector abstractions for MapReduce-based systems. Inf Sci (Ny) 2017;379:112–27. doi: https://doi.org/10.1016/j.ins.2016.08.013.
[71] Liu Y, Wei W. A Replication-Based Mechanism for Fault Tolerance in MapReduce Framework. Math Probl Eng 2015;2015. doi: https://doi.org/10.1155/2015/408921.
[72] Zhu Y, Samsudin J, Kanagavelu R, Zhang W, Wang L, Aye TT, et al. Fast Recovery MapReduce (FAR-MR) to accelerate failure recovery in big data applications. J Supercomput 2020;76:3572–88. doi: https://doi.org/10.1007/s11227-018-2716-8.
[73] Yuan Z, Wang J. Research of scheduling strategy based on fault tolerance in Hadoop platform. Commun. Comput. Inf. Sci., vol. 399 PART I, Springer Verlag; 2013, p. 509–17. https://doi.org/10.1007/978-3-642-41908-9_52.
[74] Ganglia. Ganglia Monitoring System. Ganglia 2016:1. http://ganglia.sourceforge.net/ (accessed February 28, 2020).

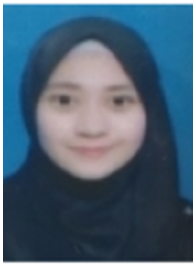
Muntadher Saadoon, core93@yahoo.com. Received BS (Hons) in Software Engineering from Al-Rafidain University College, Iraq, MS in Computer Science from University Putra Malaysia, Malaysia. He is currently a Ph.D. student in Software Engineering from University of Malaya, Malaysia. His research interests focus on software reliability and fault tolerance in distributed computing. He has strong technical knowledge in a variety of technologies, including virtualization, cloud computing, big data systems, web services, and programming languages.

Siti Hafizah Ab. Hamid, sitihafizah@um.edu.my. Received BS (Hons) in Computer Science from University of Technology, Malaysia., MS in Computer System Design from Manchester University, UK., and the Ph.D. in Computer Science from University of Malaya, Malaysia. She is currently an Associate Professor with the Department of Software Engineering, Faculty of Computer Science & Information Technology, and University of Malaya, Malaysia. She has authored over 75 research articles in different fields, including mobile cloud computing, big data, software engineering, machine learning and IoT.

Hazrina Sofian, hazrina@um.edu.my. Received Bachelor (Hons) in Computer Science, University of Malaya; Masters in Computer Science full dissertation in Requirements Engineering, University of Malaya; Ph.D. (field of study: Intelligent Computing), University Putra Malaysia. She worked for 6 years with software development industry where she gained hands-on experience in software development for banking systems, insurance systems, government systems, as well as involved in international software development project. She is currently a senior lecturer with the Department of Software Engineering, Faculty of Computer Science & Information Technology, University of Malaya. Her research interests are in distributed computing, intelligent computing, semantic web, and software requirements engineering.

Hamza H. M. Altarturi, altarturih@gmail.com. Hamza H. M. Altarturi is currently a Ph.D. student in faculty of Computer Science and Information Technology at the University of Malaya, Malaysia. He received his Bachelor of Science from Hebron University, Hebron, West Bank, Palestine in 2015. He received his Master of Computer Science from University Putra Malaysia, Malaysia, in 2017. His research interests are in software engineering, big data, and artificial intelligence.

Zati Hakim Azizul, zati@um.edu.my. Received the B.E. degree in computer science (artificial intelligence) and the M.S. degree in computer science from the University of Malaya, Malaysia, and the Ph.D. degree in computer science from the University of Auckland, New Zealand. She was a Senior Lecturer with the Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur. Her current research interests include artificial intelligence, biomedical imaging,and autonomous robot.



Nur Nasuha, nasuha@um.edu.my. Received bachelor's degree in computer science and Information Technology (Software Engineering) from University of Malaya, Malaysia. Nasuha is currently pursuing a Ph.D. program from the same university in the Department of Software Engineering, Faculty of Computer Science and Information Technology. Her PhD research is in Social network Analysis with specific focus on Link Prediction. Her research interests include social network analysis, machine learning and big data. She is also a member of IEEE student and IEEE Women in Engineering.