IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

EFFICIENT ANALYSIS IN HEALTHCARE MONITORING SYSTEM

Dr. P. Kavitha, V. Logapriya

Computer Science and Engineering,

P.S.R. Engineering college, Sivakasi, India.

Abstract: Big data systems are stable enough to store and handle large amounts of constantly changing data. Big data systems, on the other hand, are made up of large-scale hardware resources, making their subspecies prone to failure. The fundamental feature of such systems is fault tolerance, which ensures availability, reliability, and consistent performance even when there are defects. It's difficult to implement an effective fault tolerance solution in a large data system because fault tolerance must adhere to certain performance and resource restrictions. This research seeks to give a uniform understanding of fault tolerance in large data systems and to highlight typical roadblocks to fault tolerance efficiency enhancement. Previous research's fault tolerance solutions are examined to meet the stated difficulties. The report also includes a thoughtful assessment of prior research' conclusions, as well as a list of future options for addressing fault tolerance issues.

Index terms -: Fault tolerance,Fault detection Fault recovery, big data storage, big data processing

I. INTRODUCTION

Because of the increasing volume and relevance of data, the scientific community and businesses are paying close attention to big data systems [1]. Massive volumes of data are being generated by sources such as Internet of Things sensors, social media, and healthcare apps. Social media application log generation rates, for example, can reach several terabytes or petabytes per day [2]. By 2025, the International Data Corporation estimates that 163 zettabytes of data will be created [3]. Because of hardware constraints on a single server, traditional Relational Database Management Systems (RDBMS) such as MySQL [4] cannot manage the growing volume of data. To transmit huge data into the correct form, data storage and processing must be spread among trustworthy servers. Fault tolerance is an important property of big data systems because it allows for failure-free execution and prevents performance degradation [5]. As a result, decision-makers and developers regard fault tolerance as an important property of big data systems because it allows for failure-free execution and prevents performance degradation. Popular big data frameworks, such as MapReduce [6] and its open-source implementation Apache Hadoop [7], have taken fault tolerance into account by providing various fault-tolerance approaches, such as data redundancy, checkpointing, and speculative execution, which enable resiliency against failures. However, they are not always able to meet their reliability [8] and performance criteria since failures in large-scale settings have become the norm rather than the exception [9].

Fault tolerance is defined as the ability of a system to continue to function even when it encounters a fault [10,11]. The demand for efficient fault-tolerant solutions [12–20] to increase the reliability and performance of large data systems [21] has grown over time [12–20]. This requirement correlates to a rise in the number of software and hardware failures due to the underlying environment resources' scalability, complexity, and interdependency. One failure per day is predicted in a large-scale cluster of 1000 extremely dependable servers with a mean duration between failures of 30 years [5]. During the cluster's first year of operation, over 1000 individual nodes and hard disk failures occur [22,23]. The time it takes for these failures to recover might be as long as two days [24], which severely compromises performance.

On this foundation, a fault-tolerant solution for large data systems is required to improve availability, reliability, and performance in the face of failures. As a result, a literature study is required to comprehend the mechanisms by which fault tolerance has been handled in big data systems, as well as to acquire an understanding of the problems and current solutions in this field, as well as to identify new interesting research avenues. There have been few review papers focused on the issues and solutions of fault tolerance in big data systems too far. To our knowledge, only Memishi et al. [25] concentrated on fault tolerance optimization techniques in MapReduce systems from 2004 to 2016. To overcome this gap, the current work aims to analyze and examine past studies concerning fault-tolerant solutions for big data systems. This study focuses on the problems that previous researchers have experienced and the solutions that they have offered to solve those challenges.

The rest of the paper is laid out as follows. The second section gives an overview of large data systems, covering data storage and processing. Section 3 gives an overview of fault types and approaches to fault tolerance. Section 4 examines and analyzes the findings of the recommended solutions, while Section 5 classifies the fault tolerance issues and analyses the solutions mentioned in earlier research. Section 6 contains our conclusion as well as potential future study directions in this field.

1. System for Big Data:

Big data is comprised of the following elements: data, data storage, data processing, data analysis, information management, interfaces, and visualization. Data storage and processing, in particular, necessitate fault tolerance because they manage the storage and computing resources for large-scale data processing systems. These resources are also extremely vulnerable to failures due to their nature [26].

System for storing and managing data: Due to the huge amount of data being created, conventional storage systems have been limited in their ability to store and manage it. RDBMSs (relational database management systems) have historically been used to handle structured data [27]. However, these systems are incapable of storing and managing large amounts of data. To manage the massive amount of data, a scalable and reliable storage architecture that can achieve high data availability in a distributed way is necessary. There have been several storage systems suggested dealing with the problems of large data storage, including Google File System (GFS) [28], Hadoop Distributed File System (HDFS) [29], and OpenStack Swift [30]. These systems are made up of dispersed storage devices that are connected over a network and that allow virtualization, distribution, and scalability to cope with a large amount of data efficiently. Distributed storage devices are generally a network with connected storage and the ability to virtualize [31], but they may be anything. In computing, storage virtualization is a method that creates a logical representation of physical storage resources, which is represented as a signal storage pool. The network of storage devices is used to access the information stored on them, regardless of where they are or what mode they are operating in. The storage system can have three file system modes, which are file storage, block storage, and object storage, all of which are utilized in large data storage systems. File storage, block storage, and object storage are the three file system modes available.

File Storage: Data is arranged hierarchically in files, and the storage system saves all of the file information as metadata, which is then accessed by other programs. The files can be accessed by finding the path of a specific file that has been stored in the metadata.

Block Storage: Data is split into blocks, with each block containing a portion of the total amount of data. For each block, the storage system assigns a unique ID to allow applications to access the block and aggregate the blocks based on their IDs.

Object storage is a method of storing data that is wrapped with information in objects. Each item has its own set of information with specific specifications, such as its geographical location, replica number, and degree of protection, which is unique to it.

Continuing the debate on the pros and disadvantages of each file system mode, O'Reilly [32] gave a further in-depth analysis. GFS and HDFS are the most frequently used storage systems, and Verma and Pandey [33] offer a comparison of the two systems.

With the massive increase in data and the advancement of 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, as well as the development of scalable solutions and algorithms. Big data processing systems, for example, are used by data scientists for data-intensive text processing, genome assembly, machine learning and data mining, and large-scale social network research, among other applications. Large-scale data processing systems include Hadoop [7], Spark [34], Storm [35], Samza [36], and Flink [37], all of which are open-source projects under the Apache Software Foundation [38]. Hadoop is the most frequently used large data processing system, followed by Spark, Storm, Samza, and Flink. The processing engine, the cluster manager, and the processing framework are the three primary levels of these systems [39].

1. The Processing Engine, which allows for the execution of basic calculations while concealing the specifics of parallelization, distribution, load balancing, and the implementation of fault tolerance.

2. Cluster Manager: This is a centralized service that is responsible for managing the configuration of the cluster. When used in conjunction with distributed data processing frameworks, it enables dynamic resource sharing and provides effective resource management.

3. Processing Framework: A framework comprised of a collection of technologies that enables the efficient analysis of large amounts of data. A framework of this sort can enable a variety of application programming interfaces (APIs) to incorporate different types of applications and algorithms, such as scalable machine learning methods, graph-parallel computing, interactive analytics, or streaming, among others.

Big data processing systems provide a variety of characteristics that may be used in a variety of big data use cases. For example, Hadoop is capable of batch processing, while Storm and Samza are capable of stream processing, respectively. Spark and Flink, on the other hand, maybe utilized for both batch and streaming operations. When data is gathered or stored in big files, batch processing is employed to make the data more manageable. When the processing is done, the outcome of the processing, such as the creation of an index from documents for internet-scale search, must be provided. Stream processing, on the other hand, is utilized when data is continuous and must be processed quickly, such as when analyzing tweets made by users on the social networking site Twitter. A comparative analysis of the different large data processing systems was carried out by Inoubli et al. [40] as part of an experimental survey conducted by them.

3. Tolerance for failure

3.1 Different sorts of faults

Fault, error, and failure are all essential concepts in fault tolerance [41, 42] and are discussed more below. These ideas are caused by a variety of circumstances that can be produced by either software or hardware resources, and they have a variety of consequences on the behavior of the system as a whole. Figure 1 conceptually depicts the link between fault, error, and failure as they progress from the hardware to the software level, and vice versa. A fault is a non-standard hardware or software situation that causes an error to occur when the fault is in operation. An inactive fault is a circumstance in which a defect does not result in an error because it is outside of the scope of the system's capability to generate errors. Once an error is generated by an active fault, the error results in a divergence from the predicted logarithmic value, which fails the active fault. Failure is defined as the failure of a system to accomplish the purpose for which it was designed. The damage to the processor or memory, network overload, and damaged storage devices are all examples of common reasons for failure. Mistakes in software requirements or implementation, as well as external misuses, such as unexpected inputs, can all result in faults in the system being generated.

Permanent faults are distinguished from transitory faults, which are distinguished by their recurrence. A perpetual fault, also known as a fail-stop [43], is a defect that continues indefinitely and lasts for a specific amount of time. A permanent problem is straightforward to identify and locate since the defective component stays unresponsive once a permanent fault occurs [44]. This sort of fault is also easy to detect and localize. A transitory defect, which is also called a straggler [18] or a fail-stutter [45], on the other hand, makes a system accessible but with poor performance [46], as opposed to a persistent problem. In some ways, transient faults are more difficult to identify than permanent faults because they occur at a random frequency rather than at a predetermined frequency as with permanent faults; as a result, transient faults are more difficult to detect than permanent faults.

3.2 Approaches to fault tolerance

Failure tolerance is the characteristic of a system that allows it to continue operating normally even when there is a malfunction. [44] Fault-tolerant systems are built on the foundation of two fundamental concepts: fault detection and recovery. To achieve these two concepts, a variety of fault-tolerance approaches, which are classified in Fig. 2, can be used.

False-positive detection is the initial building block in the construction of a fault-tolerant system since it allows for the discovery of defects as soon as they present themselves in the system. Using heartbeat detection and fault prediction, which are both stable fault detection techniques, large-scale systems can identify and forecast faults. When error-free periods exist between two components, the heartbeat approach is designed based on an explicit and periodic exchange of heartbeat messages between the two components [44]. For example, if component A does not receive the heartbeat message from another related component B within a given period, then component A will announce that B has failed, and the fault-tolerant system will be prepared to apply a treatment action to restore functionality. Many large-scale systems, like HDFS, YARN, and Strom, have included this technique in their design and implementation. An unreliable component can only be identified if it is unable to transmit and receive heartbeat messages. Detecting problems before they arise, on the other hand, maybe extremely beneficial.

The fault prediction technique is used to solve this issue. Fault prediction is a technique that anticipates potential failures in the future by utilizing a statistical model and historical information from previously completed workloads. The statistical model is used to examine the dependencies and parameters of the workload, such as the execution time, scheduling restrictions, resource use, and machine burden before it is executed. Soualhia et al. [47] gathered historical workloads from a Google cloud cluster over one month and obtained precision up to 97.4 percent and recall up to 96.2 percent in forecasting failures using the Random Forest method. The results were published in the journal Scientific Reports. This prediction model has also been implemented in Hadoop to improve the scheduling choice for efficient fault recovery before the presence of the problem; as a result, it improves the overall reliability and performance of Hadoop [24].

Problem recovery, on the other hand, is used to restore the usual behavior of a defective component once a fault has been detected. Data redundancy techniques based on replication and/or erasure coding technologies are used in storage systems to assure data availability and dependability. In its most basic form, the replication method works by generating several copies of the original data and storing them on separate drives to provide availability and fault tolerance [48]. The GFS, HDFS, RAM Cloud [49], and Windows Azure Storage (WAS) [50] systems have all implemented this method to guarantee high data availability to their users. Once the data is transferred to the storage system, it is replicated on numerous servers and different racks, which is known as replication. The metadata file contains information on the replicas and their locations, which is accessible by the storage system. When a problem occurs, the storage system turns to the duplicated data directory for recovery when the original data files are no longer available for use. If both the original data and the replica are lost, replication becomes pointless in the worst-case scenario. Erasure coding is another kind of data redundancy that is widely used. When using erasure coding, parity data is created and stored on a separate drive with the original data, as opposed to reproducing whole blocks of data on a single disk. It is possible to recover the original data if data is lost on one disk due to a defect. Parity data is used to do this. 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, where n is the number of equal chunks in the data of size A. As a result, any k out of n would be sufficient to recreate the original data. Using the parity data, it is possible to reconstruct any two inaccessible pieces of data if the erasure coding is expressed as follows: (4, 2). Erase coding may be divided into two forms: maximum distance separable (MDS) and non-MDS. MDS is the most often utilized of the two types. The MDS erasure-coded system is capable of reconstructing every lost chunk of data based on the parity data, but the non-MDS system is only capable of reconstructing a handful of the lost chunks [32]. To improve the storage efficiency of HDFS [51], erasure coding has recently been added. Furthermore, the Reed–Solomon coding (RSC) is a common version of erasure coding that is utilized by Facebook and Microsoft storage systems [43, 52].

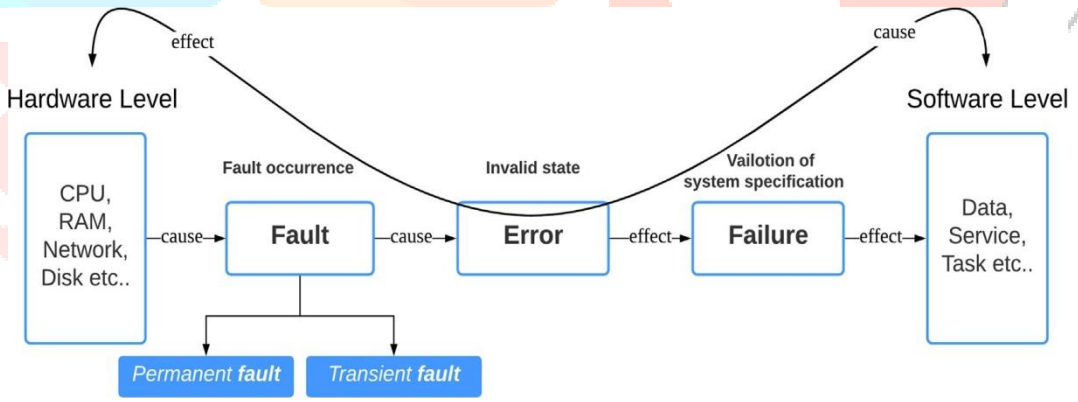

Figure 1: Relationship between fault, mistake, and failure.

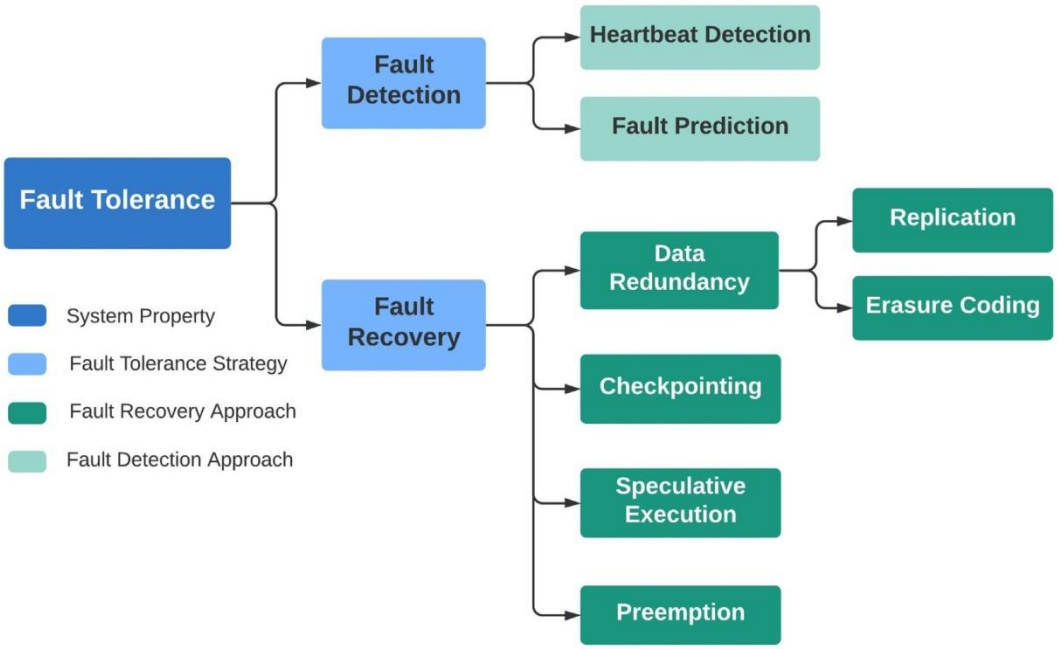

Figure 2: Approaches to fault tolerance are classified according to their fault tolerance characteristics.

Checkpointing, speculative execution and pre-emption are all examples of fault tolerance techniques that are commonly employed in large-scale data processing systems when a problem arises during the processing phase. Checkpointing works by capturing the state of an active node/process and storing it on another standby node, which is called a backup node. When an active node fails, the standby node takes over and restores the active node to its most recent recorded state, allowing for efficient and rapid fault recovery. Checkpointing is typically employed in mission-critical systems that require real-time transactions or stream processing in which latency is extremely low, such as financial systems. In the case of Flink, checkpoints are used to establish exactly-once processing guarantees by taking a snapshot of the operator state, which includes the current location of the input streams, at regular intervals [53]. Finally, speculative execution repeats the already running tasks whenever they perform worse than an acceptable level due to a suspicion of failure on the part of the user. Considering the output of the high-performing jobs and terminating their identical counterparts that are doing poorly enables for speedier fault recovery in this manner. This is the method that YARN utilizes for fault recovery. A preemption technique is also a type of fault recovery approach that is typically utilized by the task scheduler to offer efficient fault recovery when the cluster has reached its maximum resource capacity. It is necessary to have a specified task preemption strategy in place to terminate the low-priority tasks and free up the available slots to re-execute the high-priority activities if they fail.

4. Challenges and solutions for fault tolerance in large-scale data systems

Recent challenges of fault tolerance in big data storage and processing systems can be broadly classified into four categories, as illustrated in Fig. 3. These categories are storage capacity, disk I/O/bandwidth usage, fault detection latency, and fault recovery efficiency, with storage capacity being the most difficult to overcome.

5. Storage system challenges and solutions that have been presented

Failure tolerance solutions in the storage system are achieved via the use of data redundancy techniques such as replication and error-correcting coding (ECC). When it comes to replication, the storage overhead has a direct relationship with the system dependability. As a result, reducing storage overhead without compromising dependability is a significant challenge.

When compared to replication, erasure coding provides significant storage savings while maintaining reasonable dependability. However, it results in high disk I/O usage and data transmission overhead [54]. When it comes to fault tolerance in large data storage systems, the majority of the literature mentions the difficulties that must be overcome.

5.1 Delay between fault discovery and occurrence

The heartbeat detection technique [29, 62, and 63] is used in the majority of contemporary large data system implementations [29, 62, 63]. When a heartbeat is detected, a significant detection latency occurs, which means the procedure is taking longer to complete.

1) A delay happens on the static timeout value of the heartbeats, which message is received after the whole system has been impacted, and

2) A delay occurs on the static timeout value of the heartbeats

As a result, it results in a considerable performance overhead because it is the sole signal of probable permanent faults supplied while partly or temporary problems may still be capable of resisting the timer

5.2 Effectiveness of fault recovery

A failure of a single job has a detrimental impact on all of the other healthy data processing processes that are currently executing, resulting in an unpredictable execution time. There are some limits to the existing fault recovery techniques that are dependent on retry mechanisms, such as speculative execution, which requires a lot of processing and has a lot of drawbacks. When the scheduler of the resource management takes a long time to launch the recovery tasks as a result of an overloaded or full capacity of the related resources, the high computation happens. Consequently, tasks are misplaced or re-executed from the beginning as a result of this lapse in time.

5.3 Capacity for storing information

When it comes to ensuring data availability and dependability, storage systems presently rely on the replication technique. There is a significant amount of data redundancy generated by this method. For example, whenever the data is uploaded to the storage system, GFS and HDFS automatically construct three copies by default. To be more specific, holding 1 TB of data would need 3 TB of storage space, which would result in a significant rise in storage space prices as well as an increase in energy usage.

5.4 Amount of disk I/O and bandwidth used

With the advent of data redundancy and the promise of reliability assurance, erasure coding is gaining popularity owing to its potential to consume less storage space while providing a promising reliability guarantee as compared to replication [31]. According to several researchers [52,55], the reconstruction procedure of parity data stored on many drives needs more disk I/O operations than the replication method. When accessing the parity data, this issue increases the latency and the number of reading requests; it also increases the amount of data sent between the nodes, especially if the data is stored on distributed disks that are housed on several nodes in different places.

5.5 Proposed Solution:

Several researchers, including Li et al. [19,56] and Wei et al. [57], have proposed solutions for reducing the amount of replicated data while still meeting data reliability requirements. They do so by focusing on the relationship between the amount of replicated data and the requirements for data availability and reliability. Long et al. [58] and Hassan et al. [59] employ multi-objective techniques to overcome the storage capacity overhead in their respective studies. Several recent studies, including those by Huang et al. [52,60] and Sathiamoorthy et al. [43], have focused on enhancing erasure code by manipulation of local parity. It was proposed by Wei et al. [57] to use the cost-effective dynamic replication management (CDRM) strategy to achieve the lowest possible number of copies while still fulfilling the availability criterion. CDRM makes use of a mathematical model to represent the link between the number of copies and the amount of data that must be available at any one time. According to the model, the minimum number of replicas should be determined by comparing the average failure rate and the current number of replicas with the projected availability needs specified by the user. When a need is not met, CDRM dynamically produces additional copies to meet the demand. The system has been deployed and is now fully integrated with the HDFS storage system. The experimental findings demonstrated that the adaptation of this technique maintains an appropriate number of copies while maintaining a stable storage environment, which is encouraging. An alternative approach, called cost-effective dynamic incremental replication (CIR), was presented by Li et al. [56], and it is described as follows: Using a reliability model provided by the same paper, CIR dynamically decides the number of replicas to be used in the experiment. The reliability model is used to solve reliability functions that result in the smallest number of replica estimations. These functions estimate the number of copies required based on the reliability factors, such as the chance of a defect occurring and the length of time the data is stored. The functions then illustrate whether or not the current number of replicas meets the criteria for data consistency. Consequently, when the replica number does not meet the dependability requirement, additional replicas are added progressively to the system. Additionally, the study revealed that CIR significantly decreases data storage usage when data is only kept for a short period. CIR, on the other hand, is exclusively based on the reliability characteristics and price model of Amazon S3, making it inappropriate for Google clusters, which have a significantly greater failure rate than Amazon S3 storage and hence require a different storage solution. In a similar vein, Li and Yang [19] sought to reduce the number of copies while still fulfilling the criteria for data dependability, but were unsuccessful. According to the authors, a cost-effective reliability management mechanism (PRCR) is provided that is based on a generalized mathematical model that estimates data reliability in the presence of variable disk failures. When it comes to data reliability, PRCR uses a unique proactive replica checking technique to assure data dependability while retaining data with the smallest number of replicas possible. The assessment results have shown that PRCR is capable of managing a huge quantity of data while simultaneously reducing storage space usage by a substantial margin with minimal overhead.

6. Proposed options for a large data storage system

In Table 1, it is demonstrated that fault tolerance solutions offered to meet the difficulties of large data storage and processing systems may be divided into distinct solution groups based on the results of the research examined. The reliability trade-off, multi-objective optimization, and erasure coding optimization are the three primary solution categories that may be used to handle the storage system issues. To study the link between storage system data dependability and the appropriate number of replicas for lowering the storage capacity overhead, reliability trade-off methods were presented. Following that, multi-objective optimization methods have been developed to focus on improving a variety of parameters, including data availability, latency, and storage cost, that have an impact on the storage capacity overhead problem. Following that, erasure coding optimization methods based on a variety of algorithms were developed, to mitigate the problems associated with the high disk and I/O consumption in the storage system. Table 3 outlines the pros and disadvantages of each of the three solution groups in further detail.

By examining the link between the data reliability needs and the number of copies, reliability trade-off methods [19,56,57] have been found to minimize the amount of duplicated data. However, they have not much improved the situation when terabytes or petabytes of data must be replicated to ensure that data is always available. Furthermore, the studies have not taken into consideration other important aspects, such as data localization and access latency, which might negatively impact the overall performance of the storage system if the replicas are not put in the right locations. Instead, multi-objective optimization techniques [58, 59] deal with a variety of objectives, such as replica number and placement in a storage system, energy consumption, and storage costs, all of which conflict with the problem of storage capacity. They were able to obtain the optimal number of copies with the least amount of performance overhead when compared to the current replication techniques of big data storage systems, such as the Hadoop Distributed File System (HDFS). As a result of the replication-based solutions that have been presented, the amount of data redundancies in storage systems has been reduced to an absolute minimum. They are all capable of reducing storage capacity overhead while maintaining dependability, but none of them are perfect.

Erasure coding optimization methods [43, 52, and 60] were developed to improve the efficiency of erasure coding in popular storage systems, such as HDFS and WAS, by modifying the erasure coding algorithm. Huang et al. [52] were able to minimize the real cost associated with code reconstruction by 50%. During the rebuilding process, Sathiamoorthy et al. [43] were able to minimize the amount of disk I/O and network traffic. The repair overhead associated with erasure coding, on the other hand, is about ten times larger than the replication overhead. In all of the research, the focus has solely been on permanent disk defects. However, there are transitory disk faults that can also result in data being unavailable for some time. It has not yet been possible to implement a hybrid fault tolerance approach for the storage system that is based on replication and erasure coding and can provide little storage overhead while maintaining high disk I/O speed.

To solve the fault tolerance issues in the processing system, the recommended solutions from the performed research are divided into four categories: fault prediction, pre-emption, dynamic resource scaling, speculative execution, and checkpointing. Presented in Table 2 are the predicted advantages and drawbacks of each of these solution groups.

To overcome the absence of fault detection in processing systems, fault prediction solutions seek to provide early fault prediction before the occurrence of a defect. In [24, 64–67], proposed methods have decreased the number of failures and avoided an expected performance violation from occurring by detecting problems before they happen. These methods are primarily focused on permanent faults since persistent faults result in significant levels of performance overhead and resource usage, which are undesirable. Virtual big data processing environments such as Hadoop clusters and Amazon EMR are used to assess the majority of failure prediction technologies. The workload execution time and compute resource consumption are the performance assessment measures that are used to evaluate the performance of fault prediction systems in terms of fault prediction accuracy. However, even though fault prediction solutions provide early fault detection and improve system performance and resource consumption when compared to the default fault detection approach, they are still unable to predict every possible type of fault, which is especially important in large-scale environments where faults are unpredictable.

Table 1: The advantages and disadvantages of the fault tolerance methods for massive data storage systems

Categorization of Solutions	Advantages	Disadvantages
Trade-Off in Reliability	It lowers storage costs by reducing the number of copies maintained following the dependability requirement.	To achieve a balance between storage consumption and dependability, specific system settings must be specified.
Multi-Objective Optimization (MOO) is a type of optimization that considers many objectives.	It lowers storage costs by taking into account a variety of factors, including effective replica placement for the lowest latency and decreasing the number of replicas with reasonable dependability.	For a storage system that requires dynamic changes in its infrastructure, this is an inappropriate technology.
Optimization of Erasure Coding	It provides efficient data redundancy as well as a high level of fault tolerance in a single system.	Especially if the degree of data dissemination is significant, it results in high disk I/O consumption and network bandwidth overhead, which can be costly.

Table 2: The advantages and disadvantages of the fault tolerance methods for massive data processing systems

Categorization of Solutions	Advantages	Disadvantages
Prediction of Faults	It overcomes the difficulty of long detection latency by informing the user before a defect occurs.	A predictive model is used, which must be fed with different system metrics as well as faults that have been previously created.
Preemption	It makes adjustments to the task scheduler's decision by providing efficient fault recovery in terms of computation for tasks with a high level of priority.	To accomplish this, a task pre-emption strategy that takes into account task correlations, priority, execution time, and resource consumption is required.
Resource Scaling on a Dynamic Basis	It enables the system to dynamically scale up to provide a healthy environment for re-performing a defective job in the future.	It is necessary to determine the computation time of the ongoing processes on each active node to offer an appropriate resource allocation.
The practice of Speculative Execution	It achieves a high level of fault tolerance while maintaining a decent level of performance by replicating the jobs that are already running.	It necessitates the use of more computing resources.
Checkpointing	Because it records the status of each active node or job and replicates it in another stable environment, it can provide a high level of fault tolerance for real-time data processing.	Because of the difficulty in maintaining consistency of the duplicated state as well as the high cost of CPU use

Conclusions:

Providing fault-tolerant data storage and processing services is critical to the ability of big data systems to deliver dependable data storage and processing services. Big data systems are housed in large-scale settings that are susceptible to a wide range of failures. Different fault tolerance techniques have been implemented in big data systems to increase dependability and ensure that consistent data processing is maintained in the event of a failure. Fault tolerance in storage systems is achieved through the use of data redundancy techniques such as replication and erasure coding. Meanwhile, fault tolerance in the processing system necessitates the detection and recovery of errors throughout the processing process. Large-scale data systems, including the storage and processing layers, are described in detail in this work. These layers are particularly susceptible to errors and failures. Aside from that, this research offered an overview of fault tolerance ideas, covering typical fault types that occur in large-scale systems as well as fault tolerance techniques that are utilized to tolerate failures. This study also categorizes the main fault tolerance issues that have been encountered by prior studies that have offered fault-tolerant solutions for large-scale data systems. The storage capacity, disk I/O, and bandwidth use in the storage systems, as well as fault detection latency and fault recovery efficiency in the processing systems, are the problems that have been identified as classified. The techniques adopted by researchers to address each of the problems encountered in large data storage and processing systems are addressed and analyzed in greater depth. They may be used in large-scale systems such as big data systems or other similar systems to increase the efficiency of fault tolerance while also overcoming

the problems that have been identified in this paper. The conclusions of this study may be used as a guide to further research into fault tolerance for big data systems in the future, which will help to expedite the advancement of future research.

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. Secure. 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.