IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 21 January 2023, accepted 15 February 2023, date of publication 22 February 2023, date of current version 27 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3247190

RESEARCH ARTICLE

Dynamic Replication Policy on HDFS Based on Machine Learning Clustering

MOTAZ A. AHMED, MOHAMED H. KHAFAGY, MASOUD E. SHAHEEN, AND MOSTAFA R. KASEB

Department of Computer Science, Faculty of Computers and Artificial Intelligence, Fayoum University, Fayoum 63514, Egypt

Corresponding author: Motaz A. Ahmed (maa60@fayoum.edu.eg)

ABSTRACT Data growth in recent years has been swift, leading to the emergence of big data science. Distributed File Systems (DFS) are commonly used to handle big data, like Google File System (GFS), Hadoop Distributed File System (HDFS), and others. The DFS should provide the availability of data and reliability of the system in case of failure. The DFS replicates the files in different locations to provide availability and reliability. These replications consume storage space and other resources. The importance of these files differs depending on how frequently they are used in the system. So some of these files do not deserve to replicate many times because it is unimportant in the system. This paper introduces a *Dynamic Replication Policy using Machine Learning Clustering (DRPMLC)* on HDFS, which uses Machine Learning to cluster the files into different groups and apply other replication policies to each group to reduce the storage consumption, improve the read and write operations time and keep the availability and reliability of HDFS as a High-Performance Distributed Computing (HPDC).

INDEX TERMS Availability, big data, clustering, Hadoop distributed file system, high-performance distributed computing, machine learning, reliability, replication policy.

I. INTRODUCTION

The need for information storage space capacity is increasing every year due to the advent of cloud computing. Apache Hadoop is one of the most well-known frameworks in this field [1]. Apache Hadoop is developed to achieve high availability, detect and handle problems, and ensure data consistency [2]. Hadoop consists of two parts: a file system called Hadoop Distributed File System (HDFS) and a programming framework called the Map-Reduce programming model [3], [4].

Hadoop uses the Map-Reduce programming model to process big data parallel across all nodes [5]. It consists of two functions, Map and Reduce. The task was divided into sub-tasks, and the Mapper mapped each sub-task to a different node to process it in parallel. After processing each sub-task, the Reduce function will join the results of each sub-task to get the task result [6].

HDFS uses a fixed and automatic duplication mechanism to provide high-performance data accessibility. While HDFS delivers outstanding dependability, usability, and data integrity [7], its static and frequent data replication mechanism necessitates massive data storage. For example, with a default 3x replication factor [8], a file is replicated three times in various nodes [9].

The default replication assignment rule is: Place the initial copy on a random node or the home node. Suppose the HDFS client is located outside the Data nodes cluster. Then, place the next duplicate on an out-of-the-ordinary rack. Finally, the third replica is placed next to the second on the same rack as the second [10]. Choose nodes at random if there are more than three replicas. The default replica positioning policy in HDFS does not evenly distribute blocks among data nodes, resulting in discrepancies in input and output. In addition, this policy can cause specific nodes to have an excessively high IO burden while others have an abnormally low IO load. Both are incompatible with the system's overall performance [11].

The default 3x replication technique in HDFS also has a 200 % storage space and other resource overhead, such as CPU usage and network bandwidth [12]. Despite this, the replicas are only used in the event of failure. They must,

The associate editor coordinating the review of this manuscript and approving it for publication was Li Zhang.

however, be available when required. Therefore they must be live.

In 2017, erasure coding (EC) was added to Hadoop in version 3.x. Many algorithms implement erasure coding like XOR and Reed-Solomon (RS) [13], [14]. Hadoop has many policies for EC, and RS is the most used. The RS divides the data to N number of blocks and then generates M number of parity blocks [15]. If an original block of N was missed, the RS coding could recover the block through many matrices operation between M and the rest of the N blocks.

The advantage of EC is providing availability with low storage overhead. The overhead in storage is 50% of the original file size. However, low storage resources require consuming other resources such as CPU and network bandwidth [16]. Furthermore, the process of recovering the missed blocks is very complex and requires higher computational overhead [17]. Furthermore, when a client requests a file and some blocks of this file are missing, Hadoop will take more time than usual to reconstruct the whole file and send it to the client.

This paper presents a machine learning-based system to cluster the HDFS files based on their importance using unsupervised learning clustering, then selecting the suitable policy to replicate it.

The other sections of this paper are structured as follows: Section II presents related work. The proposed system is explained in section III. Section IV presents experimental results. Finally, the conclusion and future work is introduced in section V.

II. RELATED WORK

In [18], Veeraiah and Rao classifies the files into two classes, hot class and cold class, and assigns specific nodes to each class. The system increments the file replication factors and moves it to one of the hot nodes. If the system classifies the files to be in the hot class, the cold class files will have no replications and will be moved to one of the cold nodes that use EC to provide availability. This system uses an equation to calculate a threshold for file access count that determines the file class, and this strategy enhances capacity storage by up to 45%.

In [11], Swaroopa et al. Proposed a system that monitors data files in HDFS and tries classifying them into three clusters based on their importance. The clusters are hot, warm, and cold. The hot cluster should contain the most critical files on the system and set their replication factor to three. For the warm cluster, the replication factor will be reduced to two. The system will delete all file replicas for the cold cluster if the file is categorized as cold and will set the replication factor to 1. This system has no availability to the files on the cold cluster. If a client requests a file and any failure happens, the system can not find replicas for this file to respond to the client.

Yan et al. [19] proposed a dynamic replication factor based on the fault-tolerance set. DRF-FTS is a system that categorizes the files in HDFS into four types. Based on different access frequencies for the files, the system calculates the popularity of each file. The four clusters are: extremely hot, hot, warm, and cold. The system ranks the files based on their popularity, then takes the top 10% of the files to the extremely hot cluster and sets their replication factor to five. The files ranked between 10% - 20% will be set to the hot cluster with replication factor 4. If the files were ranked between 20% - 70%, the replication factor will be three. These files will be set to the warm cluster. The other files will be in the cold cluster, and the replication factor will be two. This technique aims to make load balancing for the popular files and distributes the replications across different nodes to avoid node failure.

Huang and Chen [20] uses SVR Support Vector Regression based on SVM Support Vector Machine to predict the file access and change the file replication factor based on the prediction result. After changing the replication factor, the system will distribute the replicas across the cluster nodes.

In [17], Chiniah and Mungur implements new EC policies differently than the default policies of Hadoop. RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), and RS(9,4) were implemented. The system will assign the suitable policy from the Hadoop policies and their new policies to the HDFS files according to the file size.

In [7], Kaseb et al. Introduces a Cloud Provider (CP) for the ''High Availability Redundant Independent Files'' (HARIF) system. CPHARIF manages the replications of the files manually. Unlike EC, CPHARIF does not use the RS algorithm. Instead, it splits the file into three parts with three parties using XOR operations for every possible two-part combination to provide availability and reduce storage consumption. This technique enhances storage by 66.7% compared to HDFS.

III. PROPOSED SYSTEM

The primary purpose of *DRPMLC* is to dynamically determine the replication policy for each file in the HDFS based on statistical values for the files using machine learning clustering. The system groups the files into three clusters or groups, *Hot*, *Warm*, and *Cold*, then applies different replication policies for each cluster to keep the availability and recovery for the files and improve replications size to save the storage.

Each action on HDFS files triggers saving a new record in the logs files [21]. The proposed system makes a preprocessing on these records to extract the features of each file in HDFS. The extracted features are saved in a local database to avoid recalculating features for a new file record. The machine learning model will evaluate the file using the features and determine its cluster. The system will change the replication policy of the file base on system results. If the file is clustered as a Hot file, the replication factor will be 3x as same as Hadoop default replication. The file will have two replicas if it is clustered as a Warm file, and the cold file will have no replications, but its policy will use the EC.

The proposed system reevaluates the files every week because moving the files from one cluster to another will take some time based on the current cluster of the file and its size, as shown in Table 2. Figure 1 shows the DRPMLC Architecture.

A. HDFS LOGS FILES

HDFS log files are essential because Hadoop saves all operations for all files in the log files, like reading,

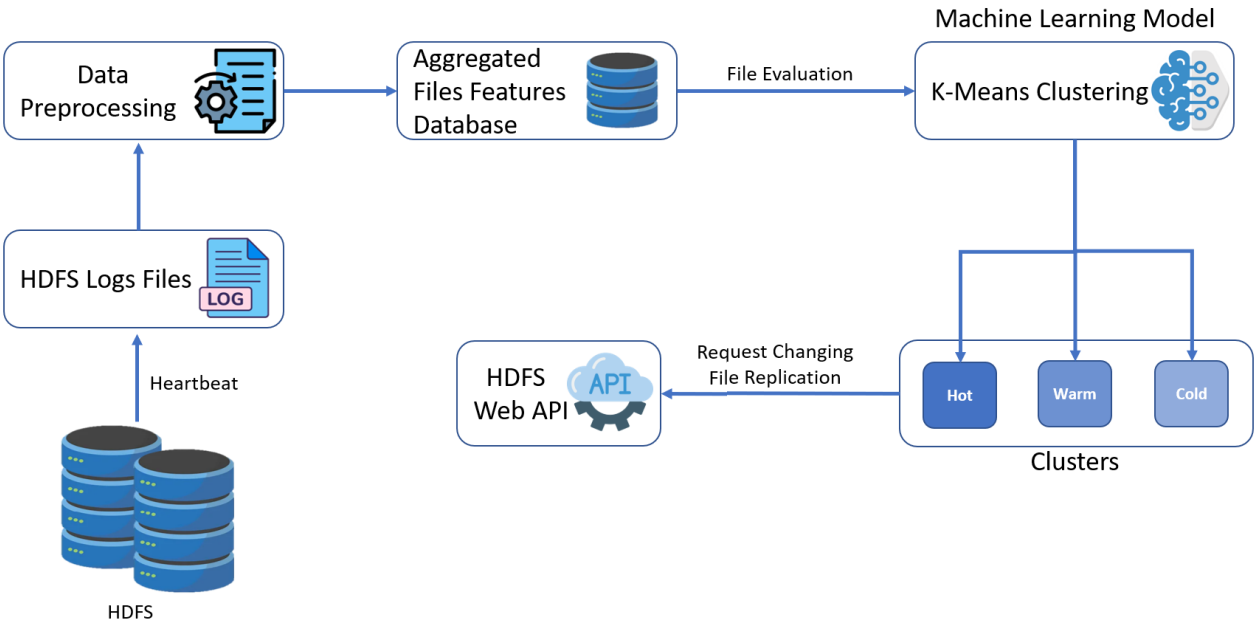

FIGURE 1. DRPMLC architecture.

creating, updating, and deleting. These files contain valuable properties for the files stored in structured records. The records structure helps us select and extract the file features and apply the preprocessing. The HDFS Audit log file contains all the details about the operations on each file, such as the date and the time of the operation, operation type, user name, user IP, file location, and file name.

B. DATA PREPROCESSING

The preprocessing stage is applied to each record in the log files to select the file features like *FileName*, *AccessDate*, and *OperationType*. Then aggregates all features for each file to extract new features like *AccessCount*, *AverageOfDateBetweenAccesses*, and *DaysFromLastAccess*. Also, the *FileSize* is an important feature, and get it for all files from the Hadoop web Application Programming Interfaces (APIs) [22]. This preprocessing is triggered when a new Record is added to the log file, so we used the Aggregated Files Features Database to save the file features results.

C. AGGREGATED FILES FEATURES DATABASE

The Aggregated Files Features Database (AFFDb) aims to facilitate extracting and saving the feature. Instead of recalculating the average, summation of the features, and processing all file records in the log files, keeping this data will optimize the next aggregation process for the values of the new feature. Each file in the HDFS has only One record in the database with many attributes that present the features. Our Algorithm shows how to select and extract features from the log file and save or update them in the *AFFDb*. After the preprocessing, the data in *AFFDb* will be used to train our ML model. Figure 2 shows the logical steps for the preprocessing and data-set construction.

Before starting the preprocessing, many operations in the log files are unimportant or not related to the file. So we have defined an array to contain only the operations that our Algorithm will process. This array is called *allowed operations array* (AOA). Then the Algorithm will loop on each record in the log file to split it and extract the values of the features. The type of operation in the record will be checked to see if it exists in AOA and will continue the processing. Otherwise, the record will be skipped and continue with moving to the next record. Suppose the operation exists in AOA. Then, we will check whether this file was processed before and exists in the AFFDb. If the file does not exist in the AFFDb, the *average of the difference between access dates* (ADBAD) for the file will be calculated in the next record of the same file using Equation (1), and the file will be added in the AFFDb.

$$ADBAD = \frac{\sum_{i=2}^{n}(AD_{(i)} - AD_{(i-1)})}{n} \tag{1}$$

where AD is the access data for the file record in the log file, and n is the access count for the file.

Instead of recalculating the ADBAD in the next file record using all values of the difference between access dates. The access date of the last record and the value of the last summation of the difference for the previous records were stored in the *AFFDb* to be used the next time to calculate the new average with the new record. First, it will be applied if the file exists in the AFFDb using Equation (2). Then the features of the file will be updated in the AFFDb.

$$ADBAD_{=} \frac{\sum_{i=2}^{n}(AD_{(i)} - AD_{(i-1)}) + AD_{(new)} - AD_{(last)}}{n+1} \tag{2}$$

where $AD_{(new)}$ is the access date of the new record in the log file, and $AD_{(last)}$ is the access data of the last record before the new record.

Algorithm 1 for Preprocessing of HDFS Logs File and Data-Set Construction

```
Input: HDFS LogsFile
Output: Aggregated Files Features Database AFFDb
define allowedOperations = {"open", "delete", "create", "append", "concat", "truncate", "rename"}
for each record ∈ LogFile do
    split record to get file features
    check the value of operation type feature in file features
    if operationType not in allowedOperations then
        move to next record
    else
        fileName=features.fileName
        if fileName not in AFFDb then
            features.AccessCount = 1
            features.FileSize = HDFS_API.getFileInfo(fileName).fileSize
            features.lastAccess = recordDateTime
            features.DaysFromLastAccess = recordDateTime - CurrentDate
            features.SumOfDifferenceBetweenAccesses = 0
            AFFDb.Add(features)
        else
            existingFile = AFFDb.selectRecord(fileName)
            existingFile.AccessCount++
            features.FileSize = HDFS_API.getFileInfo(fileName).fileSize
            SumOfDifferenceBetweenAccesses += recordDateTime - existingFile.lastAccess
            existingFile.lastAccess = recordDateTime
            AverageBetweenAccesses = SumOfDifferenceBetweenAccesses / (AccessCount-1)
            features.DaysFromLastAccess = recordDateTime - CurrentDate
            AFFDb.UpdateRecord(existingFile)
        end if
    end if
end for
```

After finishing processing all records in the log files, each file in the HDFS will have a record in the AFFDb that contains the features of the file, and the AFFDb will be the data set for training the K-Means algorithm.

D. K-MEANS CLUSTERING

Clustering is a popular statistical data analysis approach, That groups comparable items into separate groups. K-Means is the most used algorithm for data clustering, and a highly recommended algorithm for clustering big data sets [24]. K-means is the simplest among all clustering techniques and has a low execution time [25], [26], [27], [28]. It works on grouping the data into k clusters or groups. For example, in DRPMLC, the K-Means clustering algorithm is used in our model to cluster HDFS files into three clusters *Hot*, *Warm*, and *Cold*.

K-Means needs the number of clusters k as an input [29], [30], which will be three in our system, and the data set of file features. The features of a file determine which cluster will be assigned to the file. If the file has high access value, the average between each access is small, the last access is recent, and the file size is not very small. Therefore, this file is very important and should be in the *Hot* Cluster, and the system will apply the 3x replication policy to the file. The two extra replications of the file in the 3x replication policy will provide the high availability of the file. In case of any failure happens for the original file, the HDFS will have two extra copies of the file and will continue serving the requests on it normally. in this cluster, the storage for replications is still the default of Hadoop with 200% overhead. Also, if a new file were added to HDFS, it would be in the hot cluster by default and later moved to another cluster or stay in the hot cluster based on its actions.

The unimportant or dead files with low access and the average between each access are higher than the other files should be in the *Cold* cluster, and its replication policy will use *EC* RS-6-3-1024k policy. This policy has 50% overhead in storage, and the file availability will be three. However, recovering the file is very complex in the case of failure. It will consume CPU and time to recover it.

The *Warm* cluster will contain the files with medium feature values. Their importance is between the *Hot* and the *Cold* cluster. The replication policy for the *Warm* cluster is 2x replication that provides only one extra copy of the file with two availability and 100% overhead in storage.

E. APPLYING NEW REPLICATION POLICIES

Apache Hadoop is an HPDC that supports HTTP REST APIs to interact with the HDFS [22], [23], so after the system

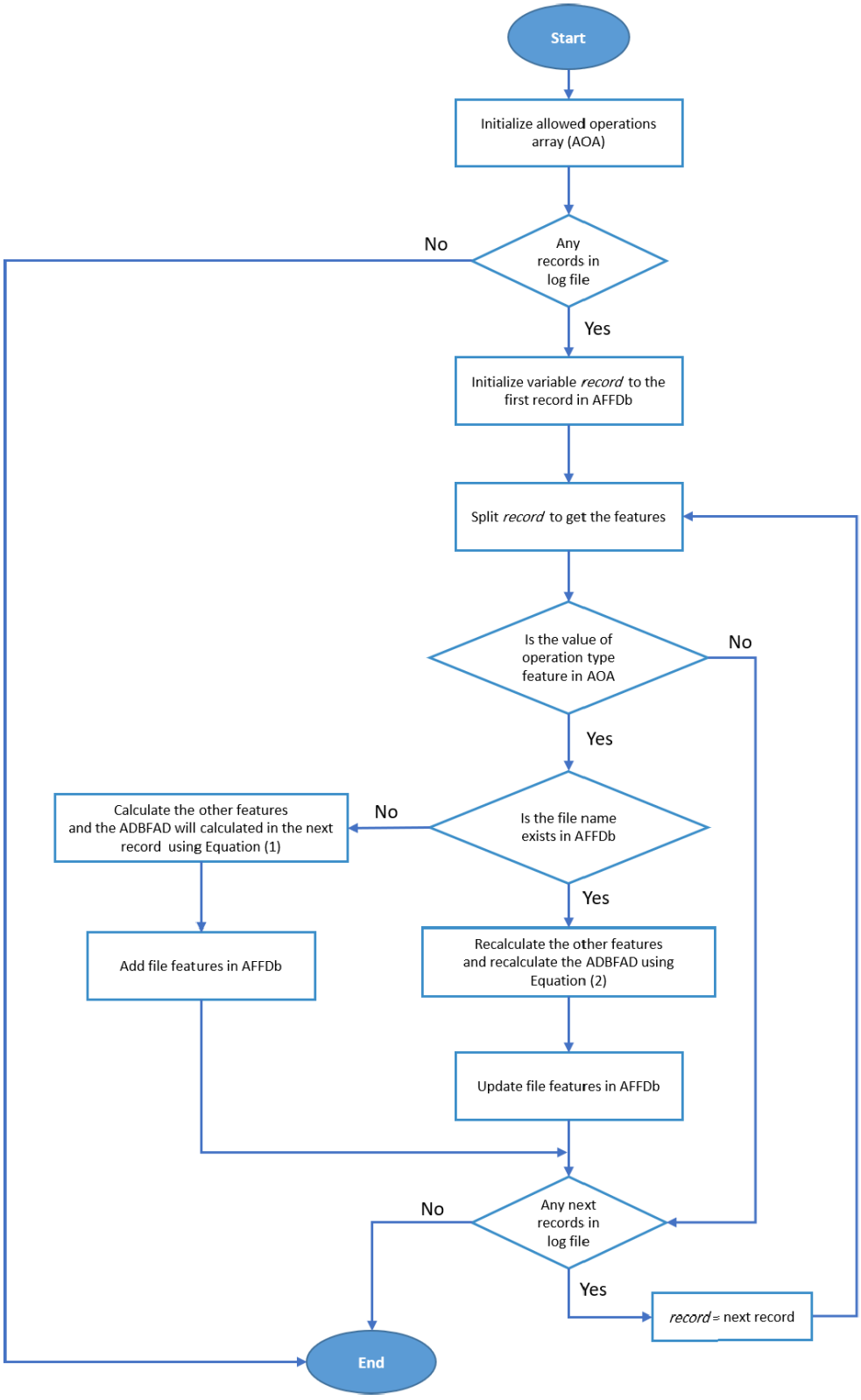

FIGURE 2. Preprocessing and data-set construction.

determines the new clusters of the files, it will call the HDFS API endpoint to apply the new replication policy for each file.

IV. EXPERIMENTAL RESULTS

In this section, we will present our experimental results in three sub-sections. Experimental setup in section IV-A,

TABLE 1. The average time of moving files between clusters.

File Size	Hot to Warm (Sec)	Hot to Cold (Sec)	Warm to Hot (Sec)	Warm to Cold (Sec)	Cold to Hot (Sec)	Cold to Warm (Sec)
128 MB	12	14	20	12	20	8
300 MB	12	21	20	14	27	45
600 MB	12	44	22	40	54	54
900 MB	12	60	32	71	114	77
1 GB	12	79	22	60	90	78
3 GB	12	208	120	217	310	192
6 GB	12	382	131	390	727	525
9 GB	12	654	193	687	939	780

TABLE 2. The average time of Read/Write operations in DRPMLC.

File Size	Cold Cluster		Warm Cluster		Hot Cluster	
	Read (Sec)	Write (Sec)	Read (Sec)	Write (Sec)	Read (Sec)	Write (Sec)
128 MB	8	10	3	4	3	14
300 MB	12	15	5	30	8	34
600 MB	21	39	30	38	29	59
900 MB	21	48	62	74	44	87
1 GB	48	52	64	64	54	90
3 GB	88	180	194	128	120	270
6 GB	226	340	336	371	198	575
9 GB	382	460	560	492	301	724

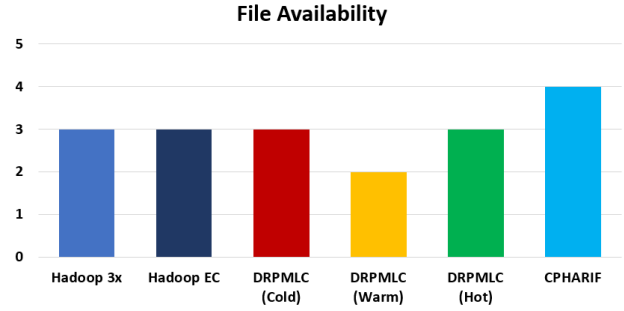

FIGURE 3. DRPMLC files availability.

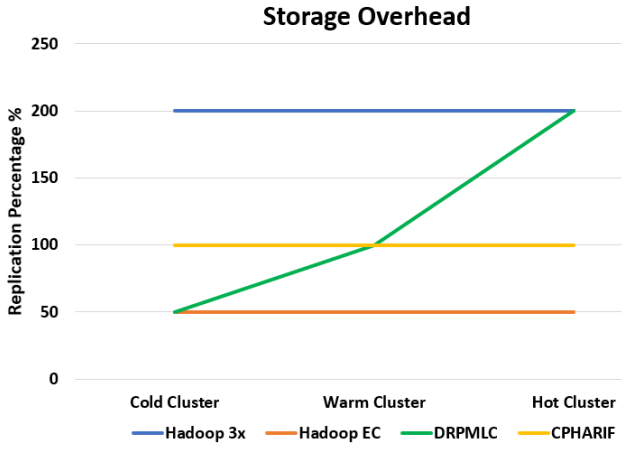

FIGURE 4. DRPMLC storage overhead.

data-set in section IV-B and the final results in section IV-C.

A. EXPERIMENTAL SETUP

DRPMLC is implemented on a private cloud, with one name node and 10 data nodes. Each node has two processors of type Intel(R) Xeon(R) Silver 4110 CPU 2.10GHz, 16 GB RAM, 500 GB SAS disks, and Hadoop 3.3.4 have been installed in all nodes. In addition, Ubuntu 20.04.2 LTS is installed on each machine.

B. DATA SET

The data set is constructed for the system based on the old logs files. First, the HDFS files were generated using TeraGen [31]. TeraGen is an official library by Hadoop to create files with different sizes in HDFS. The total size of the files is 484.85 Gigabytes, normally distributed over large, medium, and small files. Second, the actions and operations on the files were simulated and normally distributed for the simulation process to generate the log file. Finally, the log file records will be preprocessed using our Algorithm. After finishing the preprocessing, the database will contain our data set with the features of the files.

C. RESULTS

After Running the K-means for clustering the data set, *DRPMLC* clusters 100.335 GB of files as a *Hot* cluster, the *Warm* cluster 222.25 GB, and the *Cold* cluster has 162.22 GB of files. *DRPMLC* will apply the 3x replication policy for the *Hot* files as the Hadoop default 3x replication settings. Each file will have two extra copies for the replication process, so the size of replications will be the same as the Hadoop 200.67 GB and consumes 100% compared to CPHARIF Figure 5, with 200% overhead in storage too. Figure 4, and the availability of the *Hot* files is the same as Hadoop 3x, Hadoop EC, and the cold cluster, as shown in Figure 3. The *Warm* files will have only one more replication, so the replication size will be the same as the original files 222.25 GB Figure 5. The *Warm* cluster consumes the same storage as CPHARIF, and the overhead percentage is 100% Figure 4. However, Hadoop will replicate it using 3x-replication, and in this case, the replications will be 444.5 GB, and it will be available twice Figure 3. The *Cold* files will not have replications, but the erasure coding policy RS-6-3-1024k will be applied and provide the same availability. The size of parity blocks for 162.22 GB is 81.1 GB, which is 50% overhead in the storage of the files

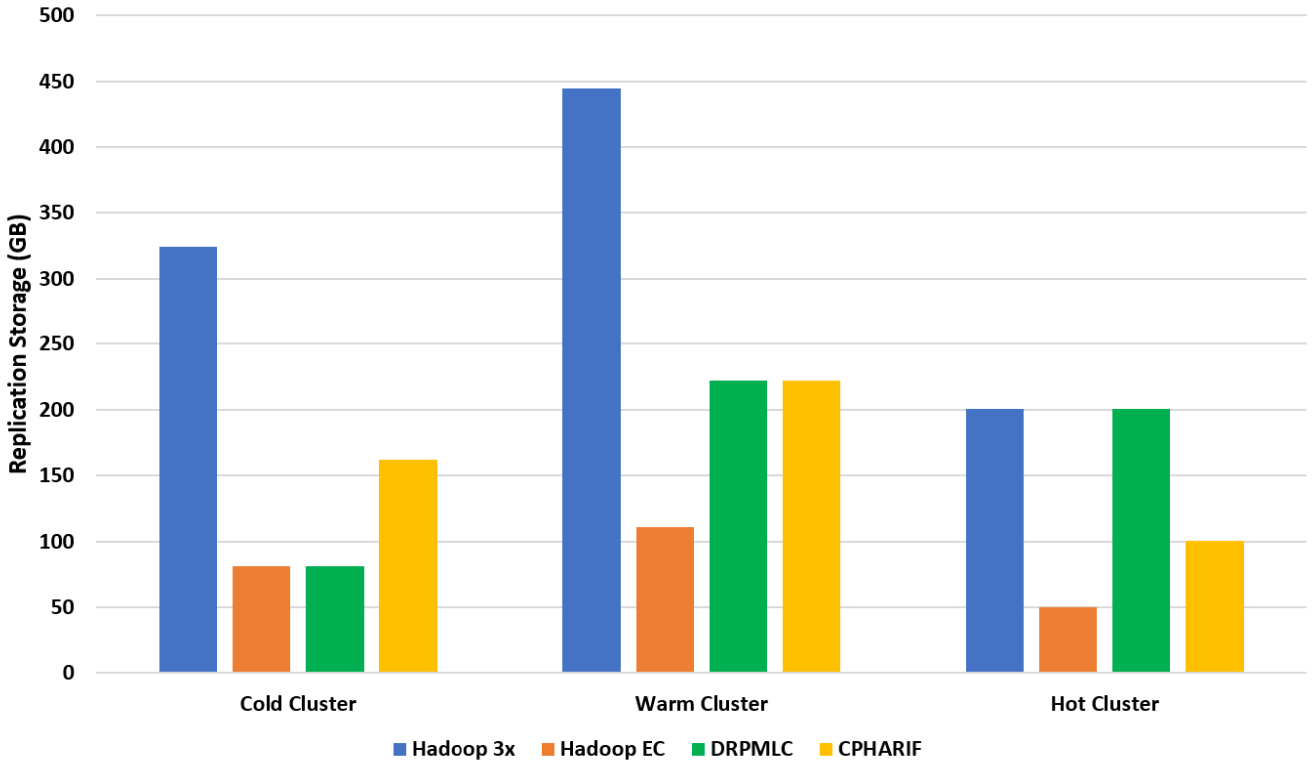

FIGURE 5. Replication Storage-based comparison between Hadoop 3x, Hadoop EC, DRPMLC and CPHARIF.

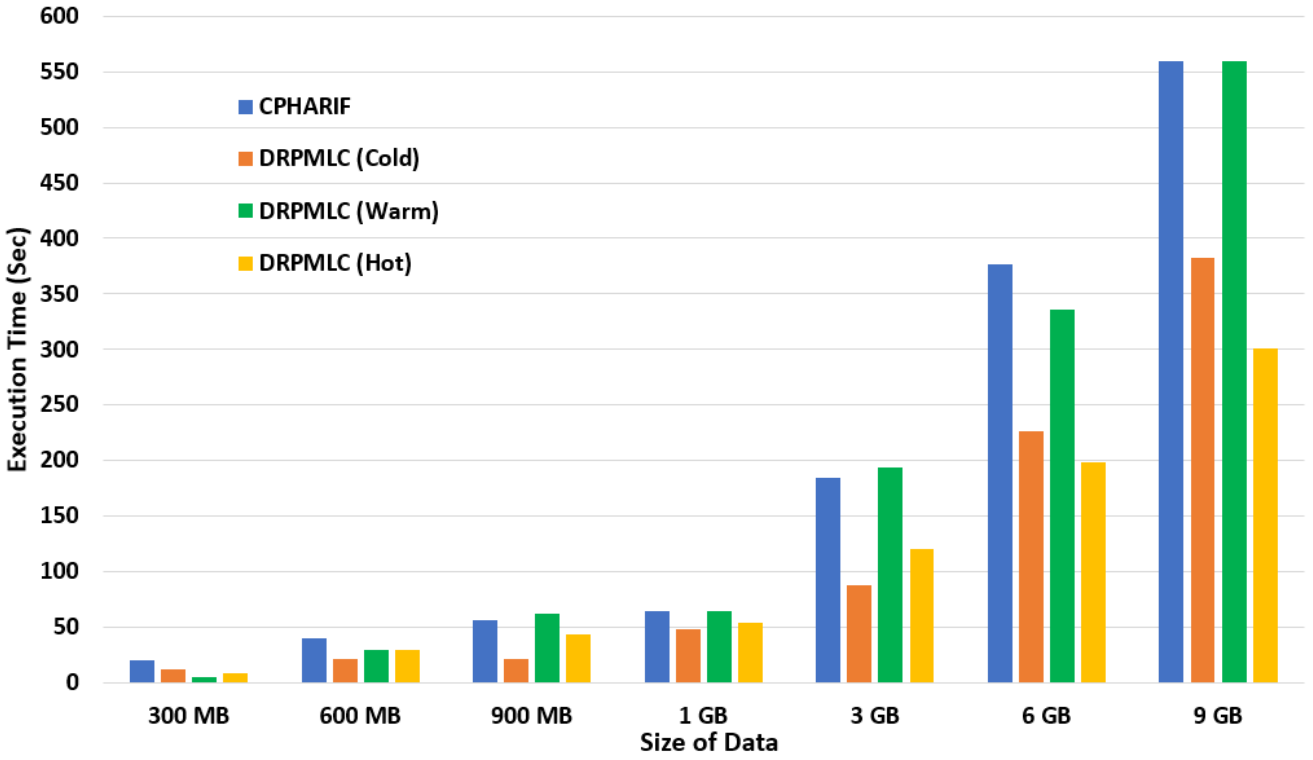

FIGURE 6. The average time of reading operation between CPHARIF and DRPMLC clusters.

Figure 4, and three availability in case of failure Figure 3. The Hadoop will replicate it twice with 324.44 GB of storage overhead Figure 5. So the total number of replications in *DRPMLC* is about 504 GB, and the total of default

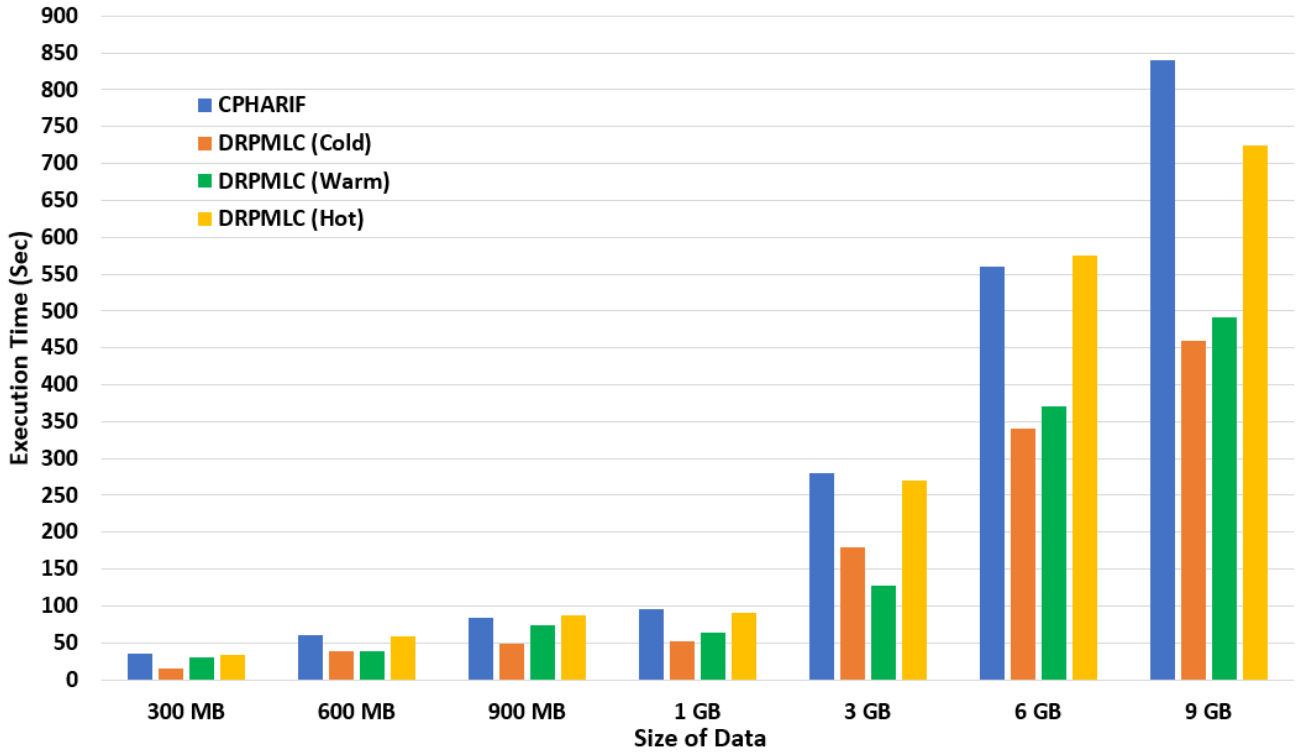

FIGURE 7. The average time of writing operation between CPHARIF and DRPMLC clusters.

replications in Hadoop is 996.7 GB. So *DRPMLC* enhances the size of the replications by about 50% of the Hadoop replications and improves the storage by 1.12% compared to CPHARIF, but CPHARIF still provides the highest availability.

Changing the file cluster consumes the time for adding or deleting one of the replicas of the file or deleting all replicas and enabling the EC policy. This overhead was measured for the default block size of 128 MB and different files with different sizes, and the experimental results show that in Table 2

Table 2 shows the average time of the Read and Write operations for Cold, Warm, and Hot clusters.

Figure 6 shows that DRPMLC has a 28.2% improvement in reading execution time compared to CPHARIF.

The average execution time to write data is improved by 29% with DRPMLC compared to CPHARIF, as shown in Figure 7.

V. CONCLUSION AND FUTURE WORK

This paper introduced a system called *DRPMLC* that clusters the HDFS files based on their importance in HDFS using machine learning clustering. *DRPMLC* has three clusters *Hot*, *Warm* and *Cold* Clusters. The system applies different replication policies to each cluster to enhance the consumed storage space of the replications and keep the files' availability as much as possible. As a result, *DRPMLC* reduced the replications space to about 50% of the default Hadoop replication size and improved the read and write operations' time respectively by 28.2% and 29% compared to CPHARIF.

In future work, the machine learning model will use many clustering techniques and another improved K-means clustering algorithm that will dynamically select the best value of k based on the data set. In addition, the system will provide many replication policies for each cluster.

REFERENCES

[1] Apache Software Foundation. (2019). *Apache Hadoop*. Accessed: Sep. 1, 2022. [Online]. Available: https://hadoop.apache.org

[2] S. Wadkar, "Hadoop concepts," in *Pro Apache Hadoop*, 2nd ed. New York, NY, USA: Apress, 2014, ch. 2, sec. 5, pp. 29–30. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4302-4864-4

[3] J. Dean and S. Ghemawat, "MapReduce," *Commun. ACM*, vol. 51, no. 1, p. 107, Jan. 2008, doi: 10.1145/1327452.1327492.

[4] M. S. Shanoda, S. A. Senbel, and M. H. Khafagy, "JOMR: Multi-join optimizer technique to enhance map-reduce job," in *Proc. 9th Int. Conf. Informat. Syst.*, Dec. 2014, pp. 80–87, doi: 10.1109/infos.2014.7036682.

[5] H. S. Abdel Azez, M. H. Khafagy, and F. A. Omara, "Optimizing join in HIVE star schema using key/facts indexing," *IETE Tech. Rev.*, vol. 35, no. 2, pp. 132–144, Feb. 2017, doi: 10.1080/02564602.2016.1260498.

[6] R. Sahal, M. Khafagy and F. A. Omara, "Big data multi-query optimisation with Apache Flink," *Int. J. Web Eng. Technol.*, vol. 13, no. 1, pp. 78–97, 2018.

[7] M. R. Kaseb, M. H. Khafagy, I. A. Ali, and E. M. Saad, "An improved technique for increasing availability in big data replication," *Future Gener. Comput. Syst.*, vol. 91, pp. 493–505, Feb. 2019, doi: 10.1016/j.future.2018.08.015.

[8] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, N. Zhang, and G. Guan, "ERMS: An elastic replication management system for HDFS," in *Proc. IEEE Int. Conf. Cluster Comput. Workshops*, Sep. 2012, pp. 32–40, doi: 10.1109/clusterw.2012.25.

[9] T. White, "MapReduce," in *Hadoop: Definitive Guide*, 3rd ed. Sebastopol, CA, USA: O'Reilly, 2012, ch. 2, sec. 5, p. 32. [Online]. Available: https://www.oreilly.com/library/view/hadoop-the-definitive/9780596521974/

[10] R. W. A. Fazul and P. P. Barcelos, "The HDFS replica placement policies: A comparative experimental investigation," in *Distributed Applications and Interoperable Systems*. Cham, Switzerland: Springer, 2022, pp. 151–166, doi: 10.1007/978-3-031-16092-9_10.
[11] K. Swaroopa, A. S. P. Kumari, N. Manne, R. Satpathy, and T. P. Kumar, "An efficient replication management system for HDFS management," *Mater. Today, Proc.*, Jul. 2021. [Online]. Available: https://www.sciencedirect.com/journal/materials-today-proceedings/articles-in-press, doi: 10.1016/j.matpr.2021.07.041.
[12] M. Saadoon, S. H. Ab. Hamid, H. Sofian, H. H. M. Altarturi, Z. H. Azizul, and N. Nasuha, "Fault tolerance in big data storage and processing systems: A review on challenges and solutions," *Ain Shams Eng. J.*, vol. 13, no. 2, Mar. 2022, Art. no. 101538, doi: 10.1016/j.asej.2021.06.024.
[13] S. B. Balaji, M. N. Krishnan, M. Vajha, V. Ramkumar, B. Sasidharan, and P. V. Kumar, "Erasure coding for distributed storage: An overview," *Sci. China Inf. Sci.*, vol. 61, no. 10, Sep. 2018, doi: 10.1007/s11432-018-9482-6.
[14] Apache Hadoop 3.3.4. *HDFS Erasure Coding*. Accessed: Jan. 5, 2023. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html
[15] L. J. Mohan, R. L. Harold, P. I. S. Caneleo, U. Parampalli, and A. Harwood, "Benchmarking the performance of Hadoop triple replication and erasure coding on a nation-wide distributed cloud," in *Proc. Int. Symp. Netw. Coding (NetCod)*, Jun. 2015, pp. 61–65, doi: 10.1109/netcod.2015.7176790.
[16] L. Xu, M. Lyu, Q. Li, L. Xie, C. Li, and Y. Xu, "SelectiveEC: Towards balanced recovery load on erasure-coded storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2386–2400, Oct. 2022, doi: 10.1109/tpds.2021.3129973.
[17] A. Chiniah and A. Mungur, "Dynamic erasure coding policy allocation (DECPA) in Hadoop 3.0," in *Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/ 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2019, pp. 29–33, doi: 10.1109/cscloud/edgecom.2019.00015.
[18] D. Veeraiah and J. N. Rao, "An efficient data duplication system based on Hadoop distributed file system," in *Proc. Int. Conf. Inventive Comput. Technol. (ICICT)*, Feb. 2020, pp. 197–200. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9112567
[19] Y. Yan, Y. Song, and B. Wang, "DRF-FTS: A dynamic replication factor replication scheme based on fault-tolerant set," in *Proc. IEEE 3rd Int. Conf. Civil Aviation Saf. Inf. Technol. (ICCASIT)*, Oct. 2021, pp. 974–979, doi: 10.1109/iccasit53235.2021.9633522.
[20] D. Huang and L. Chen, "Dynamic replica management based on SVR in IPFS," in *Proc. IEEE 14th Int. Conf. Intell. Syst. Knowl. Eng. (ISKE)*, Nov. 2019, pp. 591–597, doi: 10.1109/iske47853.2019.9170274.
[21] A. Patil, A. Wadekar, T. Gupta, R. Vijan, and F. Kazi, "Explainable LSTM model for anomaly detection in HDFS log file using layerwise relevance propagation," in *Proc. IEEE Bombay Sect. Signature Conf. (IBSSC)*, Jul. 2019, pp. 1–6, doi: 10.1109/ibssc47189.2019.8973044.
[22] *Apache Hadoop 3.3.4—WebHDFS REST API*. Accessed: Sep. 17, 2022. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/WebHDFS.html
[23] M. Elkawkagy and H. Elbeh, "High performance Hadoop distributed file system," *Int. J. Networked Distrib. Comput.*, vol. 8, no. 3, pp. 119–123, 2020.
[24] O. A. Abbas, "Comparisons between data clustering algorithms," *Int. Arab J. Inf. Technol.*, vol. 5, no. 3, pp. 1–6, Jul. 2008.
[25] K. P. Sinaga and M.-S. Yang, "Unsupervised K-means clustering algorithm," *IEEE Access*, vol. 8, pp. 80716–80727, 2020, doi: 10.1109/ACCESS.2020.2988796.
[26] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *Proc. 3rd Int. Symp. Intell. Inf. Technol. Secur. Informat.*, Apr. 2010, pp. 63–67, doi: 10.1109/IITSI.2010.74.
[27] J. Whasphuttisit, W. Jitsakul, and T. Kaewkiriya, "Comparison of clustering techniques for Thai mutual funds fee dataset," in *Proc. 14th Int. Conf. Knowl. Smart Technol. (KST)*, Jan. 2022, pp. 125–130, doi: 10.1109/KST53302.2022.9729076.
[28] Amna, N. M. Nawi, M. Aamir, and M. F. Mushtaq, "The comparative performance analysis of clustering algorithms," in *Recent Advances in Soft Computing and Data Mining*. Cham, Switzerland: Springer, 2022, pp. 341–352, doi: 10.1007/978-3-031-00828-3_34.
[29] J. A. D. Santos, T. I. Syed, M. C. Naldi, R. J. G. B. Campello, and J. Sander, "Hierarchical density-based clustering using MapReduce," *IEEE Trans. Big Data*, vol. 7, no. 1, pp. 102–114, Mar. 2021, doi: 10.1109/TBDATA.2019.2907624.
[30] H. SH. Abdallah, M. H. Khafagy, and F. A. Omara, "Case study: Spark GPU-enabled framework to control COVID-19 spread using cellphone spatio-temporal data," *Comput., Mater. Continua*, vol. 65, no. 2, pp. 1303–1320, 2020, doi: 10.32604/cmc.2020.011313.
[31] *Apache Hadoop Main 3.3.1 API*. Accessed: Sep. 20, 2022. [Online]. Available: https://hadoop.apache.org/docs/r3.3.1/api/org/apache/hadoop/examples/terasort/package-summary.html

MOTAZ A. AHMED received the B.Sc. degree in computer science from the Faculty of Computers and Information, Fayoum University, Egypt, in 2016, where he is currently a Faculty Member with the Computer Science Department, Faculty of Computers and Information. His research interests include distributed file systems, cloud computing, machine learning, and big data analysis.

MOHAMED H. KHAFAGY received the Ph.D. degree in computer science, in 2009. He worked as the Project Manager of many projects with Fayoum University, Egypt. He worked as a Postdoctoral Researcher with the DIMA Group, Technique University Berlin, in 2012. He shared to establish the first Big Data Research Group in Egypt with Cairo University, in 2013. He is currently the Dean of the Faculty of Computers and Information, Fayoum University, where he is also the Head of the Big Data Research Group. He also works with Oracle Egypt as a Consultant. He has many publications in the areas of big data, cloud computing, and database.

MASOUD E. SHAHEEN received the B.Sc. degree in science from the Department of Mathematics and Computer Science, Minia University, in 1996, the M.S. degree in computer science from the Faculty of Science, Fayoum University, Egypt, in 2005, and the Ph.D. degree in computer science from The University of Southern Mississippi, Hattiesburg, MS, USA, in 2013. He is currently the Vice-Dean of post-graduate studies and research with the Faculty of Computers and Information, Fayoum University, where he is also an Associate Professor with the Computer Science Department. He is also a Project Portal Manager with Fayoum University.

MOSTAFA R. KASEB received the B.Sc. degree in electronics and communications engineering from the Faculty of Engineering, Fayoum University, Egypt, in 2006, and the M.Sc. and Ph.D. degrees in computer engineering from the Electronics, Communications and Computer Engineering Department, Faculty of Engineering, Helwan University, in 2011 and 2019, respectively. He is currently an Assistant Dean of education and students affairs with the Faculty of Computers and Information, Fayoum University, where he is also a member of the Big Data Research Group. He is also a researcher of an international project with the west of England University (Newton-Mosharfa). His research interests include parallel and distributed systems, parallel processing, grid computing, cloud computing, database, parallel programming, business process management, and big data analysis.