IEEE Access
Multidisciplinary ⁝ Rapid Review ⁝ Open Access Journal

Received March 12, 2022, accepted March 26, 2022, date of publication March 30, 2022, date of current version April 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3163433

A Dynamic Repository Approach for Small File Management With Fast Access Time on Hadoop Cluster: Hash Based Extended Hadoop Archive

VIJAY SHANKAR SHARMA[1], ASYRAF AFTHANORHAN[2], NEMI CHAND BARWAR[3], SATYENDRA SINGH[4,5], (Member, IEEE), AND HASMAT MALIK[5,6], (Senior Member, IEEE)

[1]Department of Computer and Communication Engineering, Manipal University Jaipur, Jaipur 303007, India
[2]Faculty of Business and Management, Universiti Sultan Zainal Abidin, Gong Badak, Kuala Terengganu, Terengganu 21300, Malaysia
[3]Department of Computer Science and Engineering, MBM University, Jodhpur 342011, India
[4]School of Electrical Skills, Bhartiya Skill Development University Jaipur, Jaipur 302037, India
[5]Intelligent Prognostic Private Ltd., Delhi 110078, India
[6]BEARS, University Town, NUS Campus, Singapore 138602

Corresponding author: Hasmat Malik (hasmat.malik@gmail.com)

This work was supported by Intelligent Prognostic Private Ltd., India.

ABSTRACT Small file processing in Hadoop is one of the challenging task. The performance of the Hadoop is quite good when dealing with large files because they require lesser metadata and consume less memory. But while dealing with enormous amount of small files, metadata grows linearly and Name Node memory gets overloaded hence overall performance of the Hadoop degrades. This paper presents a dual merge technique HB-EHA (Hash Based-Extended Hadoop Archive), that will resolve the small file issue of Hadoop and provide an excellent solution for massive small files that are generated in the health care management applications. The proposed technique merges the small files using two-level compaction, therefore, the size of metadata at the name node gets reduced and less memory will be used. The indexing will be carried out over the archives and files can be accessed after merging in real-time. Index files in the proposed approach can read partially that improves the name node memory usage and also offers the file appending capability in the existing archive. The proposed technique first creates Hadoop archive from the small files and then uses two special hash functions i.e. SSHF (Scalable-Splittable Hash Function) and HT-MMPHF (Hollow Trie Monotone Minimal Perfect Hash Function), SSHF is used to dynamically distribute the archives meta-data to the associated slave index files, and these slave index files will be further written to the final index files, the order of the meta-data in final index file will be preserved by the HT-MMPHF. The evaluation outcome exhibit that the proposed technique is 13% & 17% faster than HDFS with caching enabled and disabled respectively, and 38% & 47% faster than the HAR with caching and without caching, respectively. While comparing with the map file, the proposed technique is 28 & 35 times faster with caching and without caching, respectively. HB-EHA is a maximum of 40% & 28% faster than the HBAF with and without caching, respectively.

INDEX TERMS Extended Hadoop archive, HAR archive, healthcare small files, HB-EHA, HDFS, HT-MMPHF, map file archive, sequential file, SSHF.

I. INTRODUCTION

Hadoop is a free platform for processing enormous volumes of unstructured and big data. It has a wide range of features when compared to relational databases. Using a master-slave architecture, Hadoop's file system is termed Hadoop Distributed File System (HDFS) [1]. Name node is responsible for managing the metadata of the files and have processing capabilities, it acts as a master and coordinates with slave nodes in the HDFS architecture. No processing is done on any of the data nodes that act as slaves, which means that they are just used to store data. In HDFS, files are stored in 128 MB size blocks. HDFS blocks may be whatever size the user desires; by default, it is 128 MB in size, but this

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

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

can be changed by the user. To make data availability all the time, HDFS replicates data blocks across several data nodes. To ensure the data availability data blocks are replicated on several data nodes, the value of the replication factor indicate that a single data block is written to many data nodes in the cluster and by default, the value of the replication factor is three. HDFS is well-suited to handle large files since it was designed with this in mind. Small files are generated from various application domains i.e. social media, e-commerce, online business, research and analysis, climate forecasting and educational sites, log records, health care devices and applications, etc. a file is call a small file in HDFS if it's size is less than the default block size of the HDFS. The scope of small files in technology and analytics is important and vast. Due to the excessive metadata at the name node, the overall performance of the HDFS degrades.

There are not only the processing issues with small files in HDFS but also some other issues that will significantly affect the overall performance of HDFS i.e. in case of the several file access request on the Name Node will increase the memory overheads and in case of a massive number of small files, Map Reduce Algorithm [2] requires excessive read and write operation that will take more time to process the small files in comparison with the large files. One more problem with small files is their upload time to the HDFS, the upload time of the small files is comparatively more significant than the large files. There are several feasible alternatives that Hadoop provides to solve the small file problem and its associated disadvantages i.e., HAR Archive [3], Sequential File Format [4], Combine File Input Format [5], and Map File Archive [6]. These solutions will merge the small files and merged files will be saved to the HDFS. By merging these small files, the memory overhead of the Name Node can be minimized but the access performance of the small files will degrade as the side effect of the merging. The meta-data of the small files are stored in the index files and these index files are also stored with the merged file in the HDFS. There are two major categories of the solution provided by the HDFS to solve the small file issue of Hadoop, one is the index-based archive i.e. HAR, Map File etc, and another is without indexing or based on the sequential search approach i.e. Sequential File. To access the content of a small file from the merged file, without considering the caching effect, the entire index file is imported to the memory and then the content of the small file will be recovered from the merged file with the help of meta-data written in the index file. Here meta-data will provide all the information about the small file i.e. the location of the small file in the merged file, length and size of the file, etc. Reading the entire index file in the memory will lead to excess input/output operations that result in increased access time for the small files. In case of the massive number of small files the size of the index file will be large and each time when a small file is accessed, the entire large index file will be loaded into the memory. This process will become very expensive and degrades the overall performance of the HDFS. While considering the caching effect once the index file is imported in the memory its copy will be kept in the client's memory that will improve the access performance of the archives, but the dependency of the access performance on the memory of the client is not a good practice, because client's memory is limited and not sufficient in the case of the vast number of small files.

Most of the healthcare applications/devices generate text or picture files; in our research work, we focused on text files only. Because the size of these text files is smaller than the HDFS block size, they are referred to as "small files." The proposed technique can be interfaced with the various health care devices that are generating the text files. Merging of small files is carried out to utilize the HDFS block completely. These merged small text files can be stored at the Hadoop cluster and can be accessed in real time by using the proposed approach. In this paper, a novel technique, "Hash Based-Extended Hadoop Archive (HB-EHA)" is being proposed to manage thousands of small files that are generated in the health care sector. The proposed technique's most significant feature is that it can take Hadoop archives as input and treat them as input files. The proposed technique may also retrieve the archive's meta-data directly without the usage of any caching mechanisms. HT-MMPHF (Hollow Trie Monotone Minimal Perfect Hash Function) [7]–[9] is a recommended method for locating meta-data about a file in an index file and figuring out the index file's size and location in memory. The Scalable-Splittable Hash Function (SSHF) [10]–[12] will dynamically disseminate the meta-data of small files to several index files.

II. RELATED WORK

Jude Tchaye-Kondi *et al.* [13] advocated using hash functions to create a perfect file. To obtain the metadata of a specific file, this approach eliminates the requirement to parse the whole index file. Hadoop's dynamic hashing and order-preserving operations can efficiently handle thousands of small files. Weipeng Jing *et al.* [14] suggested a dynamic queue method (DQSF) based on the analytical hierarchical process, in which small files are classified and processed according to their size. Jian-Feng Peng *et al.* [15] tackle the HDFS small file issue by merging related small files and caching frequently accessed data to reduce small file access time. By giving the notion of distribution and correlation when merging the files, Xun Cai *et al.* [16] increased the access and storage efficiency of small files. Yanfeng Lyu *et al.* [18] describe an efficient merging approach that considerably reduces the access time for small files by using the concepts of caching and prefetching. X. Fu *et al.* [19] suggested a block replica placement technique for effectively processing small files, in which files are merged according to pre-determined parameters. Qi Mu *et al.* [20] suggested an approach for dealing with small files that is both efficient and effective. The introduction of secondary indexes will increase storage efficiency and minimize Name-Node memory utilization in this system. Tao Wang *et al.* [21] suggested a method based on the behavior of small files access. To build

the association between numerous small files, the probability model is employed. Hui He *et al.* [22] provided a fantastic technique for dealing with the small file issue of the HDFS. Small files are combined in a homogenous manner by balancing the data blocks. This method improves the speed of small files in general. Songling Fu *et al.* [23] presented a novel storage system that makes use of the metadata of small files to effectively manage small files. In compared to existing competing algorithms, the experimental findings suggest that the proposed method would increase the performance of small files by a significant margin. Sharma V.S. *et al.* [24] presented a survey paper on the small file problem of the HDFS. This paper provides quick insight into the possible solution to the small file problem. Sharma V.S. *et al.* [25] evaluated various existing techniques in Hadoop that will provide the solution to the small file problem of the HDFS. N. Alange *et al.* [26] compared various existing techniques for small file problem based on the performance throughput. The CSFC technique, developed by R. Rathidevi *et al.* [27], involves grouping together small files that are connected in some way. HDFS is used for additional processing of the merged small files after clustering. Using the name node, one may store information about all of the bundled files. An innovative strategy for archiving small files was suggested by Y. Chen *et al.* [28], the logical file naming notion is used to identify the files created by the pairing at the name node. W. Wu *et al.* [29] presented a pile structure for sequential file storing that is more efficient. To combine the small files, the worst-fit approach is applied. To retrieve the merged small files from the pile structure, a global index approach is employed. R. Aggarwal *et al.* [30] explored the Hadoop small file issue in-depth and provided systematic literature as well as a Hadoop ecosystem classification. To tackle small file issue, Y. Zhai *et al.* [31] suggested a new archive file. The suggested method retrieves the meta data directly from the index file, reducing access overhead. P. Sobia *et al.* [32] undertook a thorough evaluation of the literature on particle swarm optimization (PSO) techniques for medical illness diagnosis. PSO techniques may also be utilized with a Hadoop cluster to improve response times. Bangyal W.H. *et al.* [33]–[35] suggested three unique pseudorandom initialization techniques, as well as the bat algorithm for population initialization in the PSO algorithm. The concept of pseudorandom initialization can be used to build large index files. Sharma V.S. *et al.* [36] proposed a HBAF technique that solves the small file problem by provding faster access time for small files and reducing the name node memory usage at name node, the technique proposed in this paper is the extension of the HBAF in which Hadoop archives are processed along with dual merge technique.

Table 1 summarize various existing solutions for small file problem of Hadoop. In this table, important research papers from the literature are identified and analyzed. Table 1 address the following issues: what approach is employed, what is the uniqueness of the study that is being done, and what parameters are being assessed. Table 2 shows a comparison of existing small file approaches based on a number of key parameters, including the proposed approach category, name node memory usage, whether the proposed approach can append new files or not, whether the proposed approach modifies the HDFS or not, whether an extra system is required or not, what is the amount of overheads, and what is the file access complexity of the proposed approach.

III. PROPOSED ARCHITECTURE

A. OVERVIEW OF FAST ACCESS REPOSITORY

In the present scenario, small files have applications in several domains, the application of the small files can be seen almost in every field. There are a number of applications in health care management that will generate massive small files. There is crucial requirement to store and process these small files efficiently [47], [48]. One should be able to store the small files in such a database that will provide reliable storage and faster access to the stored small files. Hadoop is the latest database technology that will provide the distributed reliable storage for the large files. Hadoop is based on the master slave technology; there is one name node and number of data nodes. Data nodes are responsible to store the data in the distributed fashion and name nodes store the metadata information of the data that is stored on the data nodes. In case of the health care management applications, there is a requirement of frequent access to the small files, if Hadoop is chosen as the database technology for health care applications, then the performance of the Hadoop degrades because there is frequent access to small files therefore name node memory usage is higher that will slow down the overall performance of the Hadoop [49]. There is a requirement of a fast access repository for Hadoop that can handle the frequent operations of the small files in health care applications. HB-EHA is a technique that will handle the frequent operations of the small files in health care management applications by merging the small files and then these merged small files are stored in the Hadoop. Hadoop treat these merged files as the larger files and processing of these merged files can be done efficiently in Hadoop. To access the small files from the merged files, HB-EHA uses two special hash functions that will locate the particular small file in the merged file and also maintain the order of the small file while appending more files to the existing archive. The whole process is depicted in figure 1.

B. PROPOSED HB-EHA APPROACH

The proposed solution HB-EHA will increase performance by providing an entry-level archive capability for small files; all small files will be archived, resulting in better memory usage and overall name node performance. Figure 2 shows how the proposed approach builds index files for small files archive's meta-data using two-level hash functions and allowing us for quick access of small files. Parallel numerous archives are formed when combining the small files. In the beginning, a temporary_master_index_file

TABLE 1. A quick review to the existing approaches for handling massive small files [24].

S. N	The outcome of the Paper	Data Management Techniques Used	The novelty of the research	Parameters for Evaluation
1.	Fast Access Container: Hadoop Perfect File (HPF) [13]	-HPF is based on the index and hash functions. -Metadata of small files are distributed in index files using the extendible hashing technique. -The data node memory is used to implement central cache management; this will reduce the disc I/O operation and provide faster access.	-Metadata of small files can be fetched partially, which will lessen the heap on the name node memory and give quicker admittance to the small files. -Append and seek operations are improved.	-Small files access time with caching and without caching -Metadata usage -Name node memory usage -Time to create a new archive -Archive file creation efficiency
2.	Technique to handle Massive Small Files [14]	-DQFS: Dynamic queues are used for the variable-sized small files. -First, the small files are merged, and then a secondary index is built over the merged small files. -To provide better access, prefetching & caching is provided	-There are different queues category for the small files in the DQSF -A logical hierarchical interaction will be utilized to assess the presentation of small files with various reaches -Dynamic queues size is also identified by the analytical model.	-Small file access time -Name Node memory consumption -Size of metadata -Time to upload data sets -Combined efficiency of DQSF
3.	Merging technique based on associated files and visiting spot [15]	-Small files are combined, and their related metadata are kept up with for the quicker access of small files. -With the utilization of the log straight model an exceptional reserve memory is planned that gives better access time to the connected little documents.	-Lower the name node memory usage -Lower the access time of random access of the small files	-Access time for the text data set -Access time for the picture data set -Memory utilization name node.
4.	Improved storage technique for small files [16]	-Small files are converged for better examination. The consolidation of files depends on the relationship and appropriation of the files (MBDC)	-Improves access and storage efficiency of the small files.	-Memory usage at Name Node -Import time of merged file -Access Time
5.	Technique to improve Small files I/O [17]	-Address variables are used for the purpose of the direct data referencing -Cache mechanism is provided for the faster access of the data blocks -The concept of inline data is used to eliminate data blocks for the zero-size files.	-Lower the disk I/O operation -Lower the latency at the application level -Accelerate overall throughput for various file operations	-Processing time (normalized) -Access time for small files -Overall throughput -Utilization of the disk
6.	Optimized technique for storing and accessing small files [18]	-Optimized merging strategy is proposed -Caching and prefetching mechanisms are used to speed up access.	-Faster access time for small files -A fast, accessible index file is built to maintain the metadata of the small files. -Lower the Name node memory usage	- Name node memory consumption -Small file access time
7.	Data replication technique in cloud environment [19]	-The accessing of small files is enhanced with the use of a block replica placement algorithm -The block replica algorithm uses an in node model that stores the small files in such a way that the access frequency of the small files can be optimized. -Small files are categorized based on their I/O frequencies and then merging of the files is carried out.	-Lower the access time of the small files -Enhance the overall data query latency and efficiency -Small files are merged that will reduce the size of metadata hence providing the faster access time -A comparative study is carried out between sequence file, HAR and D2I	-Random and sequential read/write time of the small files -Average disk usage of the slave nodes when the number of files is different in the data set -Memory usage of slave nodes
8.	Optimized technique to store small files [20]	-An intermediate layer is being inserted to improve the storage efficiency; this layer is responsible for the space requirement of the incoming small files from the client/ application. -Two-way indexing scheme is used to process the merged small files.	-Enhance the storage architecture using a secondary index mechanism. -Improves the name node memory usage and access efficiency of the storage.	-Relative comparison between uploading time v/s number of files -Relative comparison between downloading time v/s number of files
9.	Solution for HDFS small file problem [21]	-An enhanced merging technique along with prefetching is being proposed using the concept of probabilistic latent semantic analysis. -Access patterns of the small files are mined by mapping the file level accesses to the task level accesses.	-Improves the hit ratio of prefetching. -Lower the MDS workload-Memory usage of the MDS. -Lower the I/O delay (request-response) -A comparative study being carried out between the proposed PLSA model, dong model [29], Hadoop archive and native HDFS.	-Average I/O delay -Hit-ratio of prefetching -MDS memory usage and workload

TABLE 1. *(Continued.)* **A quick review to the existing approaches for handling massive small files [24].**

10.	Technique to store massive small files as per the requirement of health care application [22]	-During the merger of small files, merging queues and tolerance queues are utilized to keep track of the overall balance of data blocks. -Using a file merge queue and a tolerance queue, we may merge files according to the balance of their data blocks.	-Memory usage is reduced at all data nodes in the cluster because of the ability to balance data blocks. -Enhances the cluster's overall storage and access performance.	-Upload time of small files in HDFS. -Name node memory consumption -Processing speed of data.
11.	Enhanced file system for small files [23]	-A new file system structure is proposed that integrates different storage systems: iFlatLFS. -This flat file system uses a simple metadata management consistency scheme.	-The internet-based apps will benefit from this strategy since it improves their storage and access efficiency. -iFlatLFS can access raw disk data directly and construct the simple index file for the metadata.	-Throughput (Access/Second) -Identifying the critical point -Metadata size -File write ratio -Comparative study of iFlatLFS, Randomio, XFS, Ext4, and ReiserFS
12.	Linear Hash Function [37]	-Linear hashing function based on the extendible hashing is used to merge the small files and maintain their corresponding metadata. -When the index file is larger than the specified size, then it is split into the sublists.	-The LHF technique provides a faster access time for the handling of small files. -In this method, the size of metadata and access time is reduced. -The concept of splitting gives extra support to the algorithm.	-Storage and access efficiency -Throughput -Name node memory consumption
13.	New storage scheme for small files: SIFM [38]	-While merging the small files, correlations are considered. -Metadata file is stored at multiple nodes in a distributed fashion. -To achieve faster access, caching & prefetching mechanism is used.	-Reduce the memory footprint of the name node. -Thousands of small files can be accessed more quickly. -Reduce the latency and seek time on the disc.	-Data and name node memory usage -Small files I/O efficiency -Parallel file access efficiency.
14.	Use case of small files for weather data analytics [39]	-Sequential data structure i.e. array, is used to store the abstract path. -On the basis of the array, small files are merged, and index files are created based on the merged files.	-HDFS's small file performance is enhanced by reducing the name node's memory consumption. -Reduce the time it takes for tiny files to be uploaded.	-Name node memory consumption -Map-reduce time -Upload time -Access time
15.	Efficient technique to store and access small files [40]	-Small files are classified in the logically and structurally related files. -To access small files quickly, prefetching and caching are utilized. -To manage the metadata in local and global index file strategy is used.	-Storage efficiency is improved by the factor of 9 and 2 for the FMP-SSF and FGP-ISF respectively. -Lower the severe interaction time.	-Number of files per KB -MSPF (Milli-Second)
16.	Use case of small files for powerpoint file [41]	-Multi-level pre-fetching is used that combines the local index file and correlated files. -Small files are combined, and index files are produced locally.	-Size of metadata is reduced -Storage and access efficiency are improved using two levels prefetching	-Number of files per kilobyte -Access efficiency -Access time

and a master_name_file are created as a backup. After the slave_index_files have been constructed, this temporary_master_index_file will be deleted. The names of all the small files to be processed are retained in the HB-EHA archive in a permanent file called the master name file. Prior to being appended to a part file, small files may be compressed on the client-side, allowing for quicker processing on the client-side than on the HDFS. During the process of adding files to the part file, a capacity threshold on the size of the part file is defined and regularly checked. Once the threshold limit is reached, a new part file will be created and all other small files will be merged into that new part file. There is need to create new part files when the threshold limit gets over and the remaining small files are merged in the newly created part file. A limit on the index file's size is also required because in case of a random seek operation, a new connection is established each time to access a file from the various data nodes. A fundamental issue in our strategy is the dynamic distribution of data across many slave_index_files. This dynamic dissemination may be achieved using SSHF, and theses slave_index_files will be turned into final index files afterward. A small file is appended to a part file and its metadata and name are added to the temporary master index and master name files at the same time. The process of building final index files is accomplished in two phases, the first phase starts with the merging of small files. SSHF will be used to add the data to the corresponding slave index file. This whole process is executed on the client side. In the second stage, the slave index file is sorted. Sorting is maintained using another hash function named HT-MMPHF.

TABLE 2. Comparative study of the small file handling approaches [41].

Technique Proposed / Existing	Category	Name Node Memory Usage	Appending file support	Modification to HDFS	Requirement for the extra system	Pre-upload of HDFS required	Overheads	File access complexity
HDFS [1]	DFS	V. High	Y	-	-	Y	V. High	High
Hadoop Archive [3]	Index & Archive based	Low	N	N	N	Y	V. High	Low
Map File Archive [6]	Index & Archive based	V. Low	For special keys	N	N	N	Moderate	High(O(log_n))
Sequence File [4]	Archive based	V. Low	Y	N	N	N	Low	Low(O(n))
Blue Sky [30]	Index & Archive based	Low	Y	N	N	N	High	High
T. Zheng et al [42]	HBase and Archive based	Low	Y	Y	Y	N	High	High
New HAR [43]	Index & Archive based	Low	Y	N	N	Y	High	High
OMSS [7], TLB-Map File [44]	Sequential Map File-based	V. Low	For special keys	N	N	N	Moderate	High
SHDFS [15]	Index & Archive based	Low	Y	Y	Y	N	High	High
SFS [45]	Index & Archive based	Low	Y	Y	Y	N	High	High
Linear Hash Function [26]	Index & Archive based	Low	Y	N	N	N	Moderate	High
DQSF [14], He [22], Cai [16], Kyoungsoo [46]	Index & Archive based	Low	Y	N	N	N	High	High
Hadoop Perfect File [13]	Index & Archive based	Low	Y	N	N	N	Moderate	High(O(1))

HT-MMPHF is an order-preserving hash function that will be responsible for sorting metadata at slave_index_files. Finally, slave_index_files are written to the final_index_file.

SSHF is a hash function that belongs to the extendible hashing class [12] that allocates meta-data from small files to slave index files using a dynamic hashing approach. This arrangement involves a hash as the piece string and an arranged tree information structure for the query [50]. There are many slave_index_files that may be used for storing metadata about a given file. The last two bits of a file name hash value's bit string define the hash. All items with the identical pattern in the last bits will be included in the slave index file. Dynamic additions and deletions on slave index files are made possible by Scalable-Splittable Hashing. It is necessary to partition and generate new slave index files when an existing one has reached its limit. Because of the splitted hash procedure, it is possible to create new slave indexes dynamically. Slave_index_files might be gotten to straightforwardly during the query activity. During the production of slave_index_files and their related final_index_files, the two files are made at the same time. To ensure that the newly generated slave_index_file is in sync with the old and newly splitted slave_index_files during the split hash method, the meta data components are reorganized. It is unimaginable for an assortment of static sort 'p' keys to crash into a number kind 'q' number utilizing the HT-MMPHF hash function. q ought to be more prominent all the time than or equivalent to P, the worth of the static key. Whenever the upsides of 'q' and 'p' are equivalent, the hash function satisfies the 'minimal' condition and is alluded known as the minimal_perfect_hash_function. A minimal_ perfect_hash_function ought to be utilized to protect the request for keys; this function returns whole number qualities in the request for the static key. This approach guarantees that the lexicographic request of metadata parts in conclusive file documents is safeguarded. Metadata in slave index files is coordinated lexicographically utilizing record name hash esteems. In the final index file, the minimal _perfect_hash_function is developed and put toward the start of the hash esteems. All slave index files will be kept in touch with their final index files before the temporary master index file is erased. In comparison to other hash algorithms of similar complexity, this one's primary advantage is that it has a much lower time and space complexity (logarithmic). The time it takes to access to a given file from the final index files is very less since the meta-data elements are sorted and may be accessed instantly with (Big Oh (1)) complexity [9].

C. ACCESS TIME FOR PROPOSED HB-EHA

The access time for a particular file is the sum of time for accessing metadata and the corresponding content of the file.

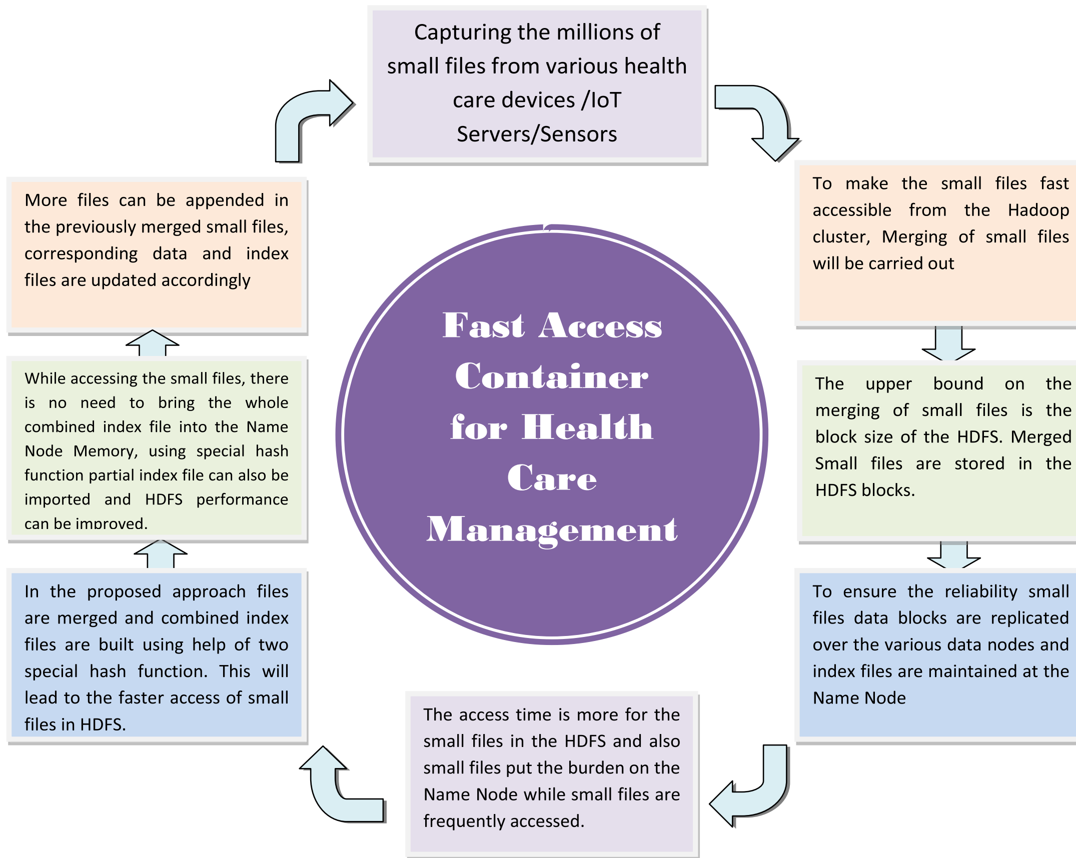

FIGURE 1. Overview of the fast access repository for health care applications.

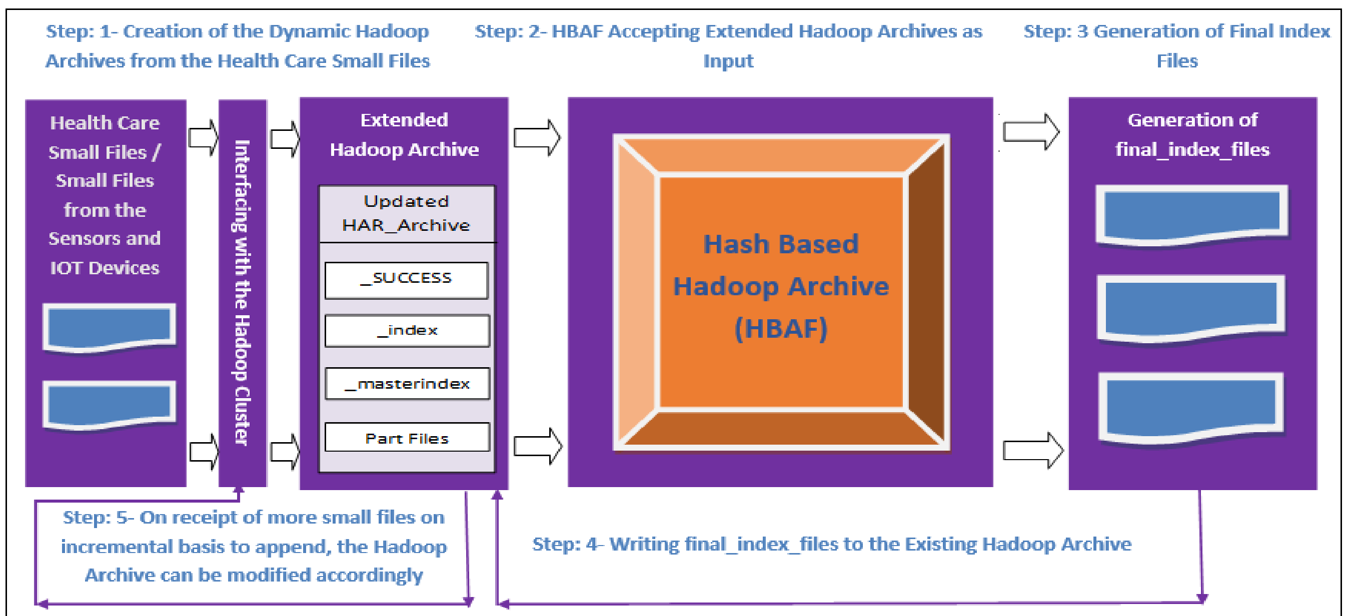

FIGURE 2. Proposed hash based extended hadoop archive (HB-EHA).

First, a client sends the file access request to the Name Node, Name Node finds out the metadata of the requested file and sends results to the client. Based on the information received by the Name Node, the client sends the request to the Data Nodes for content access.

$$T_{\text{Access-HDFS}} = T_{\text{Access-Meta}} + T_{\text{Access-Content}} \tag{1}$$

$$T_{\text{Access-Meta}} = T_{\text{Req-Meta-NN}} + T_{\text{NN-Response}} \tag{2}$$

$$T_{\text{Access-Content}} = T_{\text{Req-Read-DN}} + T_{\text{DN-Response}} \tag{3}$$

Where:

$T_{\text{Req-Meta-NN}}$ = Time to access a file and locate its metadata at Name Node

$T_{\text{NN-Response}}$ = Name Node response time to the client file access request

$T_{\text{Req-Read-DN}}$ = Time to read the desired file from the data node

$T_{\text{DN-Response}}$ = Data Node response time to the client file access request

Since HDFS provides the normal access to the files stored and does not maintain special index files for the lookup of the metadata, whereas HAR and map file maintain the special index files for the lookup of the metadata of the files and require reading and processing whole index file in the memory.

$$T_{\text{Access-HDFS}} < T_{\text{Access-HAR | Map-File}} \tag{4}$$

HAR provides two levels of indexing that maintain the metadata of the files, which results in high $T_{\text{Access}-\text{Meta}}$ in comparison to the map file.

$$T_{\text{Access-Meta (Map-File)}} < T_{\text{Access-Meta (HAR)}} \tag{5}$$

Eq. 4 states that the access time of native HDFS is lesser than the HAR and map file because these approaches process entire index files while accessing a particular small file. Eq. 5 state that the metadata access time of the map file is lesser than the HAR, because in HAR, multilevel indexing is done therefore the size of index files is more than the map file hence HAR access time is more than the map file. The content in the map file is stored sequentially therefore, to access a random file in the map file requires a very high access time in comparison to the HDFS and HAR.

$$T_{\text{Access-Content (Map-File)}} > T_{\text{Access-Content (HAR)}} > T_{\text{Access-Content (HDFS)}} \tag{6}$$

$$T_{\text{Access (Map-File)}} > T_{\text{Access (HAR)}} > T_{\text{Access (HDFS)}} \tag{7}$$

Eq. 6 states that a map file requires the highest time to access the content of a small file in comparison to the HAR and HDFS, the reason for this is the sequential access nature of the map file approach. Based on Eq. 4, 5 and 6, Eq. 7 concludes that the map file evolves the highest access time in comparison to the HAR and HDFS. The calculation of the access time of the proposed HB-EHA can be done as follows: In the proposed approach, small files are processed at two levels first, several small files are converted in the archive, and then these archives are processed using the SSHF, this two-level compaction and partial access of the index files from the archives will improve the access time of the proposed HB-EHA at a great extent. Suppose N small files are archived and stored using the Hashing based technique, then the access time for these files can be expressed as follows:

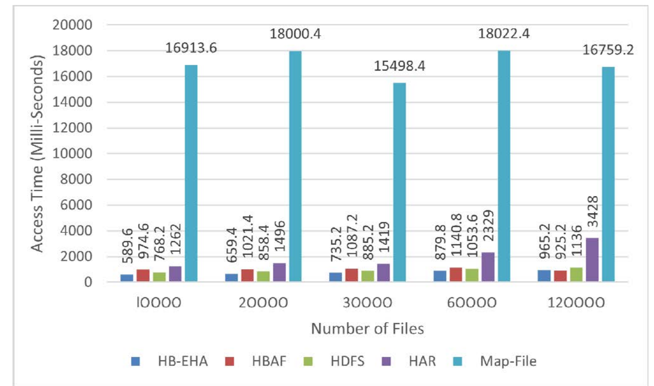

FIGURE 3. Accessing 10 files with cache enabled.

$$T_{\text{Access (HB-EHA)}} = N^{*}(T_{\text{Req-Read-DN}} + T_{\text{DN-Response}}) + \sum(T_{\text{Req-Meta-NN}} + T_{\text{NN-Response}} + T_{\text{Disk}} + T_{\text{Network}}) \tag{8}$$

The proposed HB-EHA reduces the access time in the following aspects: As the small files are merged, archived, and can be dynamically (partially) accessed, it will lead to reduce the communication time between the Name Node and client, ensure fast metadata lookup and decrease the disk overall I/O time & network latency time.

IV. EXPERIMENTAL SETUP AND RESULT ANALYSIS

A cluster of five nodes is being built up to test the proposed HB-EHA technique and other competing archives. The Name Node and Data Node have identical configurations, including an Intel®core™ i5-7500 CPU running at 3.40GHz, a 64-bit operating system, and 4 GB of RAM. The operating system in the cluster is Ubuntu 18.04.1 LTS, with open JDK-11.0.4 installed. The most recent version of Hadoop (3.1.3) is installed on all PCs connected to the Hundred MBPS (Backbone) network. The HDFS block size and RF (Replication Factor) are both set to its default values, that is 128 MB and 3, respectively. Five data sets with a variable number of text files were utilized in the testing i.e., 20000, 30000, 60000, 120000. A large number of small text files were selected for processing because, when compared to other file formats, text files provide the best results and take the least amount of time to merge and access. These file ranges between 1 KB to 1 MB in size. ten, fifty, and hundred files are randomly accessed from various archives with and without caching to assess the performance of the different archives. Without caching the performance of the Hadoop archive evolves linear and performs worst in several cases. As shown in the figures, the proposed HB-EHA performs faster than the competitive approaches. With caching, Map File performs the worst, the reason for this is the larger number of the small files and all metadata is loaded in the memory of the client. The reason for the excellent performance of the proposed HB-EHA is as follows:

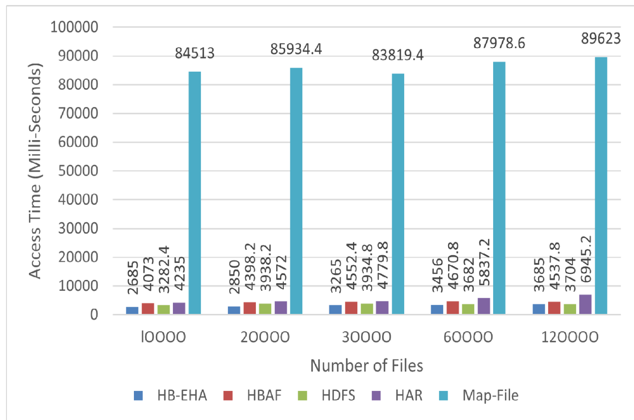

FIGURE 4. Accessing 50 files with cache enabled.

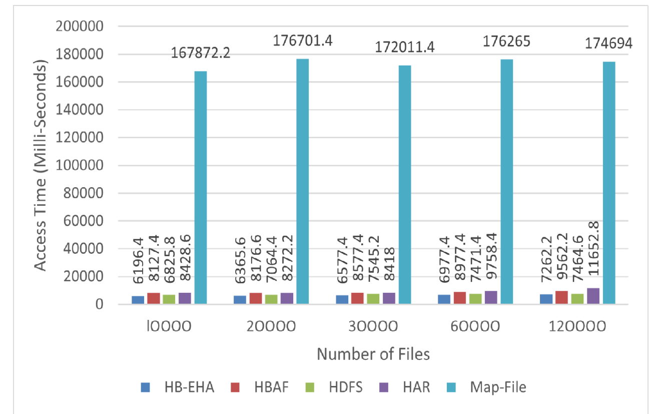

FIGURE 5. Accessing 100 files with cache enabled.

1) To access a particular file from map file and HAR, whole index file is processed therefore, more time is required to access the file, while in the case of the proposed technique, partial index files are processed.
2) In map files, small files are de-compressed at two levels, whereas in the case of the proposed technique, only one level of decompression is required.
3) For metadata storage, the proposed technique utilizes the data node's memory and relax the Hadoop main memory to facilitate the faster access of the small files.
4) In the proposed technique, files are read and written using socket communication means a client can directly interact with the data node's.

A. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 10 FILES FROM VARIOUS ARCHIVES WITH CACHE ENABLED

The proposed HB-EHA is 15 percent to 23 percent (results according to different data sets) quicker in comparison to the original HDFS and 48 percent to 71 percent quicker than the HAR_Archive. As shown in Figure 3 The map file archive will have the highest access time. The access time for HB-EHA is linear as the number of files in the health care data set grows, and it is 17 to 29 times quicker than the map_file_archive. When compared to the HBAF, HB-EHA is 23 percent to 40 percent faster with exceptional case for 120000 files. Due to the dual merge process, the performance of the HB-EHA degrades as the number of files increases in the data set.

B. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 50 FILES FROM VARIOUS ARCHIVES WITH CACHE ENABLED

The proposed HB-EHA is 1 percent to 28 percent quicker than the native HDFS and 32 percent to 46 percent quicker than the HAR_Archive. As shown in Figure 4, again map file archive will have the highest access time. HB-EHA is 24 to 31 times quicker than the map_file_archive, and access time is linear with respect to the number of files in the archive. HB-EHA is 18 percent to 35 percent faster than HBAF. HB-EHA performs relatively poor for the larger datasets in comparison to the HBAF.

C. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 100 FILES FROM VARIOUS ARCHIVES WITH CACHE ENABLED

As Figure 5 depicts, that proposed HB-EHA is 2 to 13 percent faster than native HDFS and 22 to 38 percent quicker than the HAR_Archive. HB-EHA is 24 to 28 times quicker than the map_file_archive. The access time for the health care data is linear in fashion up to some extent but when the number of files is more than lac then the slight degradation in the performance can be seen, this degradation is due to the high number of read/write operations. HB-EHA is 22 percent to 24 percent faster than HBAF, when accessing more number of files HB-EHF moves towards the stable results in comparison to the HBAF.

D. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 10 FILES FROM VARIOUS ARCHIVES WITH CACHE DISABLED

The proposed HB-EHA is 3 percent to 12 percent quicker than the native HDFS and 22 percent to 38 percent quicker than the HAR_Archive. There is an exceptional case when the number of files is 60,000, in this case, native HDFS performs 6% better than the proposed technique, and the reason for this is the real-time delay in archiving files from the health care data set. The experimental results demonstrate that map file archives take the highest time to access (except in the rare instance of HAR), whereas HB-EHA is 14 to 37 times quicker than map file. As seen in Figure 6, HAR performs the poorest with 1,20,000 small files. This unique behavior is due to HAR's multilayer indexing. While the number of files in the health care data set is in the millions, the notion of multilayer indexing will cause needless delays. HB-EHA is 10 percent to 28 percent faster than the HBAF. In case of the 20,000 dataset the performance of the HBAF and HB-EHA is almost equal and in case of the 1,20,000 data set HB-EHA is 17 percent slower than the HBAF, these are the side effects of the dual merge.

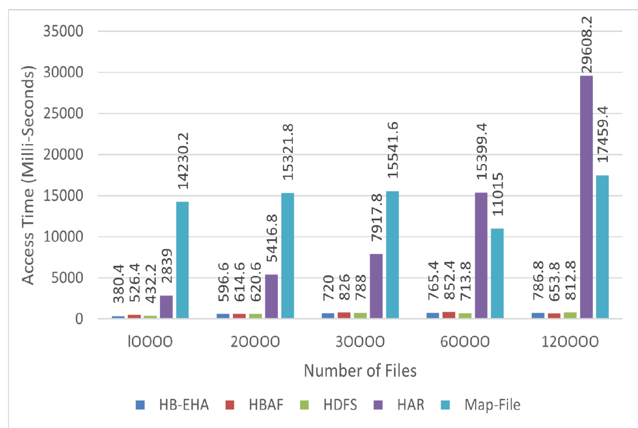

FIGURE 6. Accessing 10 files with cache disabled.

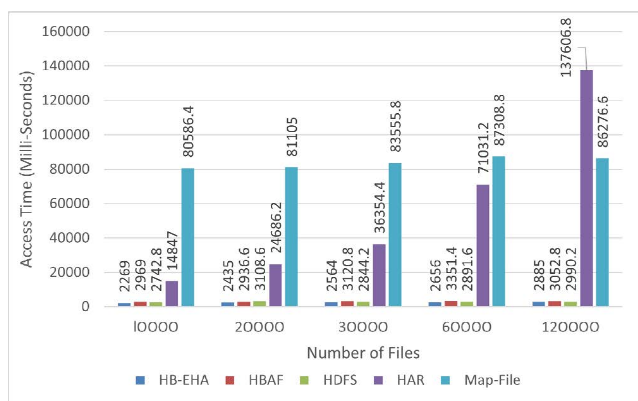

FIGURE 7. Accessing 50 files with cache disabled.

E. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 50 FILES FROM VARIOUS ARCHIVES WITH CACHE DISABLED

The proposed HB-EHA is 3 percent to 22 percent quicker than the native HDFS and 7 percent to 48 percent quicker than the HAR_Archive. The experimental results demonstrate that map_file_archive takes the highest time to access (unless in the rare instance of HAR), whereas HB-EHA is 30 to 36 times quicker than map file. Figure 7 shows that HAR performs the poorest when dealing with 120,000 small files. This unique behavior of HAR is due to a multiple indexing strategies that cause needless delays when dealing with a large number of files in a health care data set. HB-EHA is 5 percent to 24 percent faster than HBAF, HB-EHA performs well in the case of the smaller data sets.

F. EVALUATION OF ACCESS TIME FOR RANDOMLY READING 100 FILES FROM VARIOUS ARCHIVES WITH CACHE DISABLED

The proposed HB-EHA is 5 percent to 17 percent quicker than the native HDFS and 82 percent to 98 percent faster than the HAR archive. The experimental results demonstrate that the map_file_archive has the slowest access time, with HB-EHA being 31 to 35 times quicker. Figure 8 shows that the HAR has a similar uncommon situation owing to the multi-level indexing of the archive's small files. The suggested technique

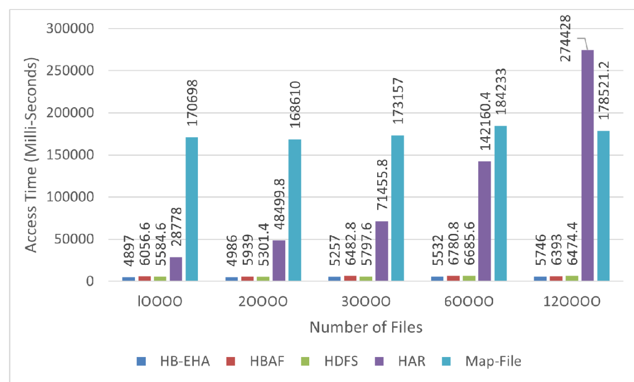

FIGURE 8. Accessing 100 files with cache disabled.

outperforms the native HDFS by a good margin. HB-EHA is 10 percent to 19 percent quicker than the HBAF.

V. CONCLUSION AND FUTURE WORK

The main motive of HDFS is to govern the ever-increasing volume of big data and ensure that the data is available and accessible at the same time. The HDFS architecture is incompatible with the small files produced by numerous application domains, such as health care, IoT devices, and sensors. In terms of retrieving small files and managing their meta-data, the HDFS is complicated. Managing small files efficiently while simultaneously lowering access time and memory usage at the name node is a pressing issue. Several scholars have researched in this topic and provided several ways for handling small files effectively. Most of the methods presented minimize name node meta-data use by transferring the indexing process to the client-side, although these techniques lag when assessed in terms of access time. There is a need for a technique that can receive small file archives as input, process them as files, and manage the index files in such a manner that the name node memory use is reduced while the small files are accessed quickly. The proposed HB-EHA works better than the HAR, Map File and HBAF. When caching is removed, the HAR's access time increases dramatically due to its multi-level indexing, but the proposed solution is unaffected by whether or not caching is enabled or disabled. The proposed solution would allow relatively quick meta-data access for small files, as well as the ability to append files after the archive has been created. For the goal of caching, data nodes are employed; this notion reduces memory load on the client-side, resulting in faster access times for small files. It will be feasible to get to the substance of small files all the more rapidly on account of the utilization of SSHF for meta-data inclusion and an order-preserving hash function (HT-MMPHF) for keeping everything under the control of meta-data inside final index files. It was demonstrated that the proposed approach is lot quicker than the HAR and map file. While caching is enabled, HB-EHA is 38 and 28 time quicker than HAR and map files, respectively. While caching is disabled, HB-EHA is 47 and 35 times quicker than the HAR and map files, respectively. When compared to the HBAF, HB-EHA is a

maximum of 40% to 28 % faster, but HB-EHA show the good performance for the smaller data sets. Due to the dual merge of the HB-EHA, the process of indexing becomes complex and HB-EHA performs poorly. The proposed approach can be more effective if a few additional features are added as future work i.e. other hash function combinations might be utilized to improve the proposed approach, client-side memory use might be diminished and it is also important to recognize the numerous parts of our approach that are yet utilizing client memory. Deletion of small files and accessing small files by the file magnitude is still a future work in our approach.

ACKNOWLEDGMENT

The authors would like to thank the Faculty of Business and Management, Universiti Sultan Zainal Abidin (UniSZA), Malaysia, for providing necessary facilities and support to this research work. They would also like to thank the Intelligent Prognostic Private Ltd., India, for providing necessary facilities and support to this research work.

REFERENCES

[1] *HDFS*. Accessed: Oct. 2, 2021. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[2] *Map Reduce Algorithm*. Accessed: Oct. 2, 2021. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[3] *HAR Archive*. Accessed: Oct. 2, 2021. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/Hadoop_archive
[4] *Sequence File*. Accessed: Oct. 2, 2021. [Online]. Available: https://examples.javacodegeeks.com/enterprisejava/apachehadoop/hadoop-sequence-file-example
[5] *CombineFileInputFormat*. Accessed: Oct. 2, 2021. [Online]. Available: https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/lib/input/CombineFileInputFormat.html
[6] *MapFile*. Accessed: Oct. 2, 2021. [Online]. Available: http://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/io/MapFile.html
[7] S. Sheoran, D. Sethia, and H. Saran, "Optimized MapFile based storage of small files in Hadoop," in *Proc. 17th IEEE/ACM CCGRID*, Madrid, Spain, May 2017, pp. 906–912.
[8] D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna, "Theory and practice of monotone minimal perfect hashing," *ACM J. Exp. Algorithmics*, vol. 16, p. 26, May 2011.
[9] D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna, "Monotone minimal perfect hashing: Searching a sorted table with O(1) accesses," in *Proc. SODA*, Jan. 2009, pp. 785–794.
[10] G. Mendelson, S. Vargaftik, K. Barabash, D. H. Lorenz, I. Keslassy, and A. Orda, "AnchorHash: A scalable consistent hash," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 517–528, Apr. 2021.
[11] K. Claessen and M. H. Pałka, "Splittable pseudorandom number generators using cryptographic hashing," in *Proc. Haskell*, New York, NY, USA, 2013, pp. 47–58.
[12] D. Zhang, Y. Manolopoulo, Y. Theodoridis, and V. J. Tsotras, "Extendible hashing," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. New York, NY, USA: Springer, 2018.
[13] J. Tchaye-Kondi, Y. Zhai, K.-J. Lin, W. Tao, and K. Yang, "Hadoop perfect file: A fast access container for small files with direct in disc metadata access," 2019, *arXiv:1903.05838*.
[14] W. Jing, D. Tong, G. Chen, C. Zhao, and L. Zhu, "An optimized method of HDFS for massive small files storage," *Comput. Sci. Inf. Syst.*, vol. 15, no. 3, pp. 533–548, 2018.
[15] J.-F. Peng, W.-G. Wei, H.-M. Zhao, Q.-Y. Dai, G.-Y. Xie, J. Cai, and K.-J. He, "Hadoop massive small file merging technology based on visiting hot-spot and associated file optimization," in *Proc. 9th Int. Conf. BICS*, Xi'an, China, 2018, pp. 517–524.
[16] X. Cai, C. Chen, and Y. Liang, "An optimization strategy of massive small files storage based on HDFS," in *Proc. JIAET*, 2018, pp. 225–230.
[17] H. Kim and H. Yeom, "Improving small file I/O performance for massive digital archives," in *Proc. IEEE 13th Int. Conf. e-Sci. (e-Science)*, Oct. 2017, pp. 256–265.
[18] Y. Lyu, X. Fan, and K. Liu, "An optimized strategy for small files storing and accessing in HDFS," in *Proc. IEEE Int. Conf. CSE, IEEE Int. Conf. EUC*, Jul. 2017, pp. 611–614.
[19] X. Fu, W. Liu, Y. Cang, X. Gong, and S. Deng, "Optimized data replication for small files in cloud storage systems," *Math. Problems Eng.*, vol. 2016, pp. 1–8, Dec. 2016.
[20] Q. Mu, Y. Jia, and B. Luo, "The optimization scheme research of small files storage based on HDFS," in *Proc. 8th Int. Symp. Comput. Intell. Design*, Dec. 2015, pp. 431–434.
[21] T. Wang, S. Yao, Z. Xu, L. Xiong, X. Gu, and X. Yang, "An effective strategy for improving small file problem in distributed file system," in *Proc. 2nd Int. Conf. Inf. Sci. Control Eng.*, Apr. 2015, pp. 122–126.
[22] H. He, Z. Du, W. Zhang, and A. Chen, "Optimization strategy of Hadoop small file storage for big data in healthcare," *J. Supercomput.*, vol. 72, no. 10, pp. 3696–3707, Aug. 2016.
[23] S. Fu, L. He, C. Huang, X. Liao, and K. Li, "Performance optimization for managing massive numbers of small files in distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3433–3448, Dec. 2015.
[24] V. S. Sharma and N. C. Barwar, "Data management techniques in Hadoop framework for handling small files: A survey," in *Information Management and Machine Intelligence* (Algorithms for Intelligent Systems). Singapore: Springer, 2019, pp. 425–438.
[25] V. S. Sharma and N. C. Barwar, "Performance evaluation of merging techniques for handling small size files in HDFS," in *Data Analytics and Management* (Lecture Notes on Data Engineering and Communications Technologies), vol. 54. Singapore: Springer, 2021, pp. 137–150.
[26] N. Alange and A. Mathur, "Small sized file storage problems in Hadoop distributed file system," in *Proc. Int. Conf. Smart Syst. Inventive Technol.*, Nov. 2019, pp. 1202–1206.
[27] R. Rathidevi and R. Parameswari, "Performance analysis of small files in HDFS using clustering small files based on centroid algorithm," in *Proc. 4th Int. Conf. I-SMAC*, Oct. 2020, pp. 640–643.
[28] Y. Chen, J. Zhang, Z. Wang, G. Liao, S. Liu, H. Tan, G. Yang, Y. Fang, S. Wang, and Z. Sun, "A faster read and less storage algorithm for small files on Hadoop," in *Proc. Int. Conf. Comput. Eng. Artif. Intell. (ICCEAI)*, Aug. 2021, pp. 206–210.
[29] W. Wu, H. Liu, L. Duan, and S. Xu, "SequenceFile storage optimization method based on pile structure," in *Proc. IEEE Int. Conf. Artif. Intell. Comput. Appl. (ICAICA)*, Jun. 2021, pp. 118–122.
[30] R. Aggarwal, J. Verma, and M. Siwach, "Small files' problem in Hadoop: A systematic literature review," *J. King Saud Univ., Comput. Inf. Sci.*, Sep. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821002585?via%3Dihub, doi: 10.1016/j.jksuci.2021.09.007.
[31] Y. Zhai, J. Tchaye-Kondi, K.-J. Lin, L. Zhu, W. Tao, X. Du, and M. Guizani, "Hadoop perfect file: A fast and memory-efficient metadata access archive file to face small files problem in HDFS," *J. Parallel Distrib. Comput.*, vol. 156, pp. 119–130, Oct. 2021.
[32] S. Pervaiz, Z. Ul-Qayyum, W. H. Bangyal, L. Gao, and J. Ahmad, "A systematic literature review on particle swarm optimization techniques for medical diseases detection," *Comput. Math. Methods Med.*, vol. 2021, pp. 1–10, Sep. 2021.
[33] W. H. Bangyal, K. Nisar, A. A. B. Ag. Ibrahim, M. R. Haque, J. J. P. C. Rodrigues, and D. B. Rawat, "Comparative analysis of low discrepancy sequence-based initialization approaches using population-based algorithms for solving the global optimization problems," *Appl. Sci.*, vol. 11, no. 16, p. 7591, Aug. 2021.
[34] H. T. Rauf, J. Ahmed, and W. H. Bangyal, "A modified bat algorithm with torus walk for solving global optimisation problems," *Int. J. Bio-Inspired Comput.*, vol. 15, no. 1, p. 1, 2020, doi: 10.1504/IJBIC.2020.10027535.
[35] W. H. Bangyal, A. Hameed, J. Ahmad, K. Nisar, M. R. Haque, A. A. A. Ibrahim, J. J. P. C. Rodrigues, M. A. Khan, D. B. Rawat, and R. Etengu, "New modified controlled bat algorithm for numerical optimization problem," *Comput., Mater. Continua*, vol. 70, no. 2, pp. 2241–2259, 2022, doi: 10.32604/cmc.2022.017789.
[36] V. S. Sharma and N. C. Barwar, *A Novel Technique for Handling Small File Problem of HDFS: Hash Based Archive File (HBAF)*. Amsterdam, The Netherlands: IOS Press, 2021, doi: 10.3233/APC210205.
[37] W. Tao, Y. Zhai, and J. Tchaye-Kondi, "LHF: A new archive based approach to accelerate massive small files access performance in HDFS," in *Proc. 5th IEEE Int. Conf. Big Data Service Appl.*, Apr. 2019, pp. 40–48.
[38] Y. Mao, B. Jia, W. Min, and J. Wang, "Optimization scheme for small files storage based on Hadoop distributed file system," *Int. J. Database Theory Appl.*, vol. 8, no. 5, pp. 241–254, Oct. 2015.

[39] M. S. G. Prasad, H. R. Nagesh, and M. Deepthi, "Improving the performance of processing for small files in Hadoop: A case study of weather data analytics," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 5, 2014, Art. no. 64366439.
[40] B. Dong, Q. Zheng, F. Tian, K.-M. Cao, R. Ma, and R. Anane, "An optimized approach for storing and accessing small files on cloud storage," *J. Netw. Comput. Appl.*, vol. 35, no. 6, pp. 1847–1862, Jun. 2012.
[41] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li, "A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by powerpoint files," in *Proc. IEEE Int. Conf. Services Comput.*, Jul. 2010, pp. 65–72.
[42] T. Zheng, G. Fan, and W. Guo, "A method to improve the performance for storing massive small files in Hadoop," in *Proc. 7th Int. Conf. Comput. Eng. Netw.*, Jul. 2017, p. 22.
[43] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," in *Proc. 11th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, May 2014, pp. 200–205.
[44] B. Meng, W.-B. Guo, G.-S. Fan, and N.-W. Qian, "A novel approach for efficient accessing of small files in HDFS: TLB-MapFile," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD)*, May 2016, pp. 681–686.
[45] Y. Huo, Z. Wang, X. Zeng, Y. Yang, W. Li, and Z. Cheng, "SFS: A massive small file processing middleware in Hadoop," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–4.
[46] K. Bok, H. Oh, J. Lim, Y. Pae, H. Choi, B. Lee, and J. Yoo, "An efficient distributed caching for accessing small files in HDFS," *Cluster Comput.*, vol. 20, no. 4, pp. 3579–3592, Dec. 2017.
[47] S. Bahri, N. Zoghlami, M. Abed, and J. M. R. Tavares, "Big data for healthcare: A survey," *IEEE Access*, vol. 7, pp. 7397–7408, 2019.
[48] S. Nazir, S. Khan, H. U. Khan, S. Ali, I. Garcia-Magarino, R. B. Atan, and M. Nawaz, "A comprehensive analysis of healthcare big data management, analytics and scientific programming," *IEEE Access*, vol. 8, pp. 95714–95733, 2020.
[49] P. K. Sahoo, S. K. Mohapatra, and S.-L. Wu, "Analyzing healthcare big data with prediction for future health condition," *IEEE Access*, vol. 4, pp. 9786–9799, 2016.
[50] R. E. Tarjan and R. F. Werneck, "Dynamic trees in practice," in *Experimental Algorithms* (Lecture Notes in Computer Science), vol. 4525, C. Demetrescu, Ed. Berlin, Germany: Springer, 2007, pp. 80–93.

VIJAY SHANKAR SHARMA received the B.E., M.E., and Ph.D. degrees from the MBM Engineering College, Jodhpur, which one of the oldest engineering college of India. He is currently working as an Assistant Professor (Senior Scale) with the Department of Computer and Communication Engineering, Manipal University Jaipur, India. He has teaching experience of more than ten years. He has been published 11 research papers in international and national journals/conferences. His research interests include networking and simulation, big data analytics, Hadoop, and theory of computation.

ASYRAF AFTHANORHAN received the dual Bachelor of Science degree in statistics from Universiti Teknologi Mara (UiTM), in 2009 and 2012, respectively, the Master of Science degree in mathematical science from Universiti Malaysia Terengganu, in 2013, and the Ph.D. degree in management statistics from Universiti Sultan Zainal Abidin (UniSZA), Terengganu, Malaysia, in 2017. Before he pursues his Ph.D. degree, in 2015, he was a Political Researcher under Malaysia Chinese Association (MCA) for one year and five months. He is currently a Professor with UniSZA. He is appointed as the Editor-in-Chief of *The Journal of Management Theory and Practice* (UniSZA). He has been working as a Senior Lecturer with UniSZA, since October 2017. At present, he received numerous awards from innovation research and publication. He has been a member of International Association of Engineers (IAENG), Elsevier Advisory Panel, and Publons Academy Peer Review, since 2020. He is often invited as a Speaker or a Facilitator for the Scopus publication, structural equation modeling and statistics workshop. He has published more than 100 of research papers and two academic books related to tourism, transportation, management, marketing, hospitality, and statistical modeling. His profile can be referred from Google Scholar, ORCID ID, Scopus, and Google Knowledge Panel.

NEMI CHAND BARWAR received the B.E. degree in computer technology from MANIT Bhopal, the M.E. degree in digital communication, and the Ph.D. degree from the MBM Engineering College, Jodhpur, India. He is currently a Professor and the Head of the Department of Computer Science and Engineering, MBM University. He is also working as a Professor and the Head of the Department of Computer Science and Engineering, MBM Engineering College, JNV University, Jodhpur. He has experience of over 27 years in the field of teaching and research. He has published more than 50 research papers in national and international conferences and journals. He is supervising the Ph.D. Research Program in computer science and engineering discipline and into information technology. His research interests include computer networking, WSN, MANET/VANET, the IoT, big data analytics, VoD, P2P networks, and machine learning. He had organized ten national conferences and short term courses sponsored by AICTE/UGC/DST. He is a Life Member of ISTE and IEI.

SATYENDRA SINGH (Member, IEEE) received the bachelor's degree (B.E.) in electrical engineering from the Government Engineering College Bikaner, Rajasthan, India, in 2008, the master's degree in power systems from the National Institute of Technology (NIT), Hamirpur, Himachal Pradesh, India, in 2011, and the Ph.D. degree in electrical engineering from the Malaviya National Institute of Technology (MNIT), Jaipur, India, in 2019. He is currently working as an Assistant Professor with the School of Electrical Skills, Bhartiya Skill Development University Jaipur, Rajasthan. His research interests include power systems, power system economics, electricity market, renewable energy modeling, FACTs devices, multi-agent systems, and nature-inspired algorithms.

HASMAT MALIK (Senior Member, IEEE) received the Diploma in electrical engineering from Aryabhatt Government Polytechnic Delhi, the B.Tech. degree in electrical engineering from Guru Gobind Singh Indraprastha University, Delhi, the M.Tech. degree in electrical engineering from the National Institute of Technology (NIT) Hamirpur, India, and the Ph.D. degree in electrical engineering from the Indian Institute of Technology (IIT), Delhi.

He was worked as an Assistant Professor for more than five years with the Division of Instrumentation and Control Engineering, Netaji Subhas Institute of Technology (NSIT), Dwarka, Delhi, India. He is currently a Chartered Engineer (C.Eng.) and a Professional Engineer (P.Eng.). He is also Postdoctoral Researcher with Berkeley Education Alliance for Research in Singapore (BEARS) (a Research Center, University of California at Berkeley, Berkeley, USA), University Town, NUS Singapore, since January, 2019. He has published widely in international journals and conferences his research findings related to intelligent data analytics, artificial intelligence, machine learning applications in power system, power apparatus, smart building and automation, smart grid, forecasting, prediction, and renewable energy sources. He has authored and coauthored more than 100 research papers, nine books and 15 chapters, published by IEEE, Springer, and Elsevier. He has supervised 25 PG students. His research interests include the application of artificial intelligence, machine learning and big-data analytics for renewable energy, smart building & automation, condition monitoring, and online fault detection & diagnosis (FDD).

Dr. Malik is also a fellow of the Institution of Electronics and Telecommunication Engineering (IETE) and a member of the Institution of Engineering and Technology (IET), U.K., the Computer Science Teachers Association (CSTA), the Association for Computing Machinery (ACM) EIG. He is a Life Member of the Indian Society for Technical Education (ISTE), the Institution of Engineers (IEI), India, the International Society for Research and Development (ISRD), London, and Mir Laboratories, Asia. He received the POSOCO Power System Award (PPSA-2017) for his Ph.D. work for research and innovation in the area of power systems. He also received the best research papers awards from IEEE INDICON-2015, and the Full Registration Fee Award from IEEE SSD-2012, Germany.