International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-1, October, 2019

File Formats for Big Data Storage Systems

Samiya Khan, Mansaf Alam

Abstract*: Big data is one of the most influential technologies of the modern era. However, in order to support maturity of big data systems, development and sustenance of heterogeneous environments is requires. This, in turn, requires integration of technologies as well as concepts. Computing and storage are the two core components of any big data system. With that said, big data storage needs to communicate with the execution engine and other processing and visualization technologies to create a comprehensive solution. This brings the facet of big data file formats into picture. This paper classifies available big data file formats into five categories namely text-based, row-based, column-based, in-memory and data storage services. It also compares the advantages, shortcomings and possible use cases of available big data file formats for Hadoop, which is the foundation for most big data computing technologies. Lastly, it provides a discussion on tradeoffs that must be considered while choosing a file format for a big data system, providing a framework for creation for file format selection criteria.

***Keywords*: Big Data Storage, File Formats, Hadoop, Storage Systems**

I. INTRODUCTION

Development of big data systems requires identification, integration and performance optimization of several tools, technologies and concepts. Sustainability of such heterogeneous environments requires data sharing between multiple processes, which can be performed using two possible methods. The first method makes use of APIs provided by independent solutions to retrieve data from them. For instance, if Drill [1] and HBase [2] are used together, Drill shall use the HBase API for data reads. Although, data sharing can be performed in real-time using this approach, it suffers from some potential issues [3].

The overheads for deserialization and serialization are very high and may cause serious bottleneck issues for applications such as workflows. Moreover, there is no standardization in how the data should be represented. Developers perform custom integrations using existing or customized systems. In order to manage these issues, another method for data sharing may be used, which enables sharing by means of common file formats that are optimized for high performance in analytical systems [4]. Owing to this, big data file formats form an important facet of big data storage optimization.

Hadoop [5] offers one of the most cost-effective and efficient ways to store data in huge amounts. Moreover, structured, semi-structured and unstructured data types can be stored and then, processed using tools like Pig [6], Hive [7] and Spark [8] to gain the results required for any future analysis or visualization. Hadoop allows the user to store data on its repository in several ways. Some of the commonly available and understandable working formats include XML [9], CSV [10] and JSON [11].

Although, JSON, XML and CSV are human-readable formats, but they are not the best way, to store data, on a Hadoop cluster. In fact, in some cases, storage of data in such raw formats may prove to be highly inefficient. Moreover, parallel storage is not possible for data stored in such formats. In view of the fact that storage efficiency and parallelism are the two leading advantages of using Hadoop, the use of raw file formats may just defeat the whole purpose.

CERN [12] had chosen four candidate technologies, ORC [13], Parquet [14], Kudu [15] and Avro [16] for this purpose. However, we have included other data file formats like Arrow [17] and text-based formats to this discussion for comprehensibility. Fig. 1 shows the classification and hierarchy of big data file formats. Each of these file formats has their own sets of advantages and disadvantages.

Existing literature discuss data formats for specific application areas like bioinformatics [4], recommender systems [13] and their usages in satellite image detection [18] or indoor air quality analysis [19]. Some papers provide an account of how compression can be performed using some formats like Parquet and Avro [20] and performance analysis of use-case with and without the use of these formats [17]. However, there is no dedicated analysis of big data file formats. Moreover, none of the available studies provide a generalization of use cases that are most applicable for specific file formats.

This paper explores different aspects of big data file formats and shall provide a discussion on the criteria that must be used for selecting a file format. The contributions of this paper include (1) it provides a classification of available file formats for big data systems (2) it provides a comparison and use-case analysis of the available file formats (3) it provides a discussion on the aspects that must be considered while choosing a file format for a big data system.

The rest of the paper is organized in the following manner: Section 2 describes the different file formats shown in Fig. 1. Section 3 provides a comparison and use-case analysis of the available big data file formats. Section 4 discusses the different factors and tradeoffs that must be considered before choosing a file format for a big data system. Finally, the paper concludes in Section 5 and provides insights for future work.

II. BIG DATA FILE FORMATS

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Samiya Khan*, Department of Computer Science, Jamia Millia Islamia, New Delhi, India. Email: samiyashaukat@yahoo.com
Mansaf Alam*, Department of Computer Science, Jamia Millia Islamia, New Delhi, India. Email: malam2@jmi.ac.in

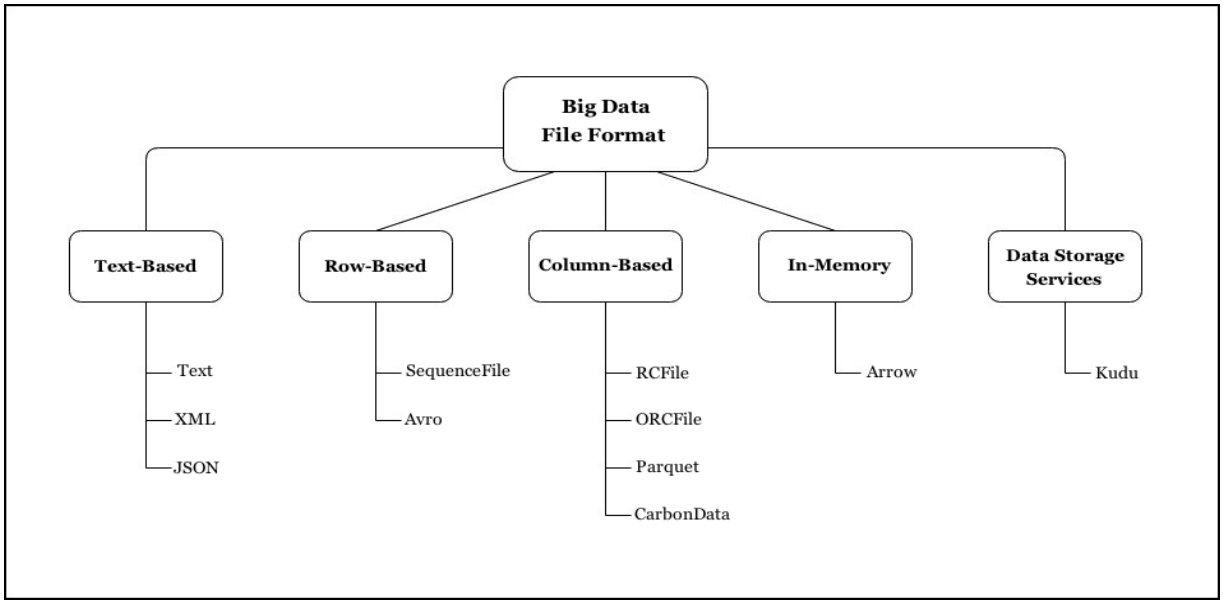

Fig. 1. Classification of Big Data File Formats

A. Text-Based Formats

Simplest example of such a data file format is entries stored line-by-line terminated by a newline character or carriage return.

In order to compress data, a file-level compression codec like BZIP2 [21] needs to be used. Three formats namely text or CSV [10], JSON [11] and XML [9] are available under this category

- Plain Text

Data stored in this format is mostly formatted in such a manner that fields either has fixed width or are delimiter-separated entries, as is the case of CSV in which entries are separated using commas.

- XML

It is possible to use external schema definitions in XML. However, the performance of serialization and deserialization is usually poor.

- JSON

JavaScript Object Notation (JSON) [22] is more performance effective than XML, but the problem with serialization and deserialization performance exists.

B. Row-Based Formats

Different types of row-based big data files formats are as follows –

- SequenceFile

This file format is supported by Hadoop framework as part of which a large file has container of binary key-value pairs for storing multiple files of small size. Therefore, the key-value pairs corresponding to records are encoded. The integration of SequenceFile [23] data file format is smooth with Hadoop in view of the fact that the former was developed for the latter.

- Avro

Apache Avro [16] is used for compact binary format as a data serialization standard. Typically, it is used to store persistent data related to communication protocols and HDFS [24]. One of the major advantages of using Apache Avro is high ingestion performance, which is attributed to fast and lightweight deserialization and serialization. It is important to mention that Avro does not have an internal index. However, the directory-based partitioning technique available in HDFS can be used for facilitating random access of data. Data compression algorithms supported by Apache Avro include DEFLATE [25] and Snappy [19].

There are other serialization and deserialization frameworks like Thrift and Protocol buffers [26] that stand in competition with Avro. However, Avro is a built-in component of Hadoop while these frameworks are external. Besides this, schema definition in Avro is done using JSON whereas Thrift and Protocol buffers depend on Interface Definition Languages (IDLs) [26] for defining the schema.

C. Column-Based Formats

Different types of column-based big data files formats are as follows –

- Record Columnar (RC) File Format

This is the most primitive columnar record format that was created as part of the Apache Hive project. RCFile [27] is a binary format similar to SequenceFile that assures high compression for operations that involve multiple rows. Columns are stored as a record in a columnar fashion by creating row splits. Vertical partitions are created on row splits in a columnar manner. The metadata is stored for each row split as the key and the corresponding data is kept as its value.

- Optimized Row Columnar (ORC) File Format

Optimized Row Columnar [13] is similar to Parquet and RCFile in the sense that all these three data file formats exist within the Hadoop framework. The read performance is better for ORC. However, writes are slower than average. Besides this, the encoding and compression capabilities of this file format are better than that of its counterparts, with ORC supporting Zlib [28] and Snappy [19].

- Parquet

Apache Parquet [14] is a data serialization standard, which is column-oriented and known to improve the efficiency significantly. This data format also includes optimizations like compression on series of values that belong to the same column resulting in improved compaction ratios. Besides this, encodings like bit packing, dictionary and run length encoding are additional optimizations available. In order to compress data, Snappy [19] and GZip [29] algorithms are supported.

- CarbonData

This data format was developed by Huawei to manage existing shortcomings in already available formats. CarbonData [30] is a relatively new data file format that allows developers to reap the benefits of column-oriented storage while also providing support for handling random access queries. Data is grouped into blocklets, which is stored alongside other information about the data like schema, indices and offsets, in addition to others. Metadata is stored in headers and footers, which offers significant performance optimization during scanning and processing of subsequent queries.

D. In-Memory Formats

Apache Arrow [17] is a platform for development of applications using in-memory data. Moreover, it works across languages, which makes it a standard for columnar memory format, enabling support for hierarchical as well as flat data. The data is organized to provide high performance analytics on modern hardware. Interprocess communication and streaming works on zero-copy or no deserialization and serialization. Besides this, it provides many computational libraries for advanced complexities.

E. Data Storage Services

Kudu [15] is a storage system that ensures scalability and is based on the concept of distributed storage. Data is stored inside tables. Moreover, this file format achieves optimized trade-off between performance and speed of ingestion, which is attributed to the columnar data organization and maintenance of index. Data compression algorithms supported by Apache Kudu include LZ4 [31], Zlib [28] and Snappy [19].

III. Comparison of Big Data File Formats

The actual storage of data on a file system is determined by the chosen data file format, which is a crucial choice to make in view of the fact that it plays a significant role in optimizing system storage. The decision of choosing a file format for an application depends on the use-case or the algorithm being used for data processing. With that said, the chosen file format must fulfill some basic requirements to be deemed appropriate for big data systems.

Firstly, the chosen big data file format must be expressive and well defined. Secondly, it should have diverse handling capabilities as far as supported data structures are concerned. Some of the basic structures that must be supported include structs, maps, numbers, strings, records and arrays, to name a few. Finally, the big data file format must be binary, simple and provide support for compression.

One of the biggest bottleneck issues in HDFS-related applications that make use of technologies like Spark and Hadoop include reduction in time taken by data reads and writes. Issues like storage constraints, evolving schemas and big data further complicate the system requirements. In order to mitigate these issues across application domains and problem scenarios, several big data file formats have come into being. The use of an appropriate file format can benefit the system in following ways –

1. Read time is reduced.
2. Write time is reduced.
3. The files can be split, which in other words means that there is no longer the need to read the whole file for retrieving a smaller sub-section.
4. There is support for schema evolution and schema can be changed on request depending upon the changing needs of the system.
5. There is availability of advanced compression codecs to ensure that files can be compressed without losing the advantages offered by the base format.

In view of the above-mentioned advantages, choosing the right data file format can optimize system performance substantially. However, a plethora of options are available in this regard. While some file formats are developed for general use, there are some others that offer optimization benefits to specific applications or improve specific characteristics. This makes a comparison of the file formats essential for facilitating the decision of which data file format is best suited for an application. Table I summarizes the advantages, disadvantages and typical use cases for the different data file formats discussed in the previous section.

IV. DISCUSSION

It can be inferred from the comparison table that even though text-based formats are simple and lightweight, they present a host of drawbacks that can affect the performance of the system considerably. In order to overcome the limitations of text-based formats, Hadoop has inbuilt data file formats. The first of these formats is a row-based data file format called SequenceFile [23]. Other data file formats that are based on SequenceFile and are included in the Hadoop ecosystem include MapFile, SetFile and BloomMapFile [32]. These file formats are designed for specialized use cases. SequenceFile only supports Java, which makes it language-dependent, and does not support versioning. As a result, Avro [16], which overpowers SequenceFile, has come up as the most popular row-based data file format. Understandably, Avro is the best option among row-based data file formats.

Table I – Comparative Analysis of Available Big Data File Formats

Class	Data File Format	Advantages	Disadvantages	Matching Use Cases
Text-Based Data File Formats	Text XML [9] JSON [11]	1. Light weight	1. Read is slow. 2. Write is slow. 3. Space is wasted because of column headers that are not required. 4. Compressed files cannot be split, which leads to huge maps. Moreover, block-level compression is not supported.	Appropriate for starting use of structured data on HDFS. Also, CSV files are used in cases where data needs to be extracted from Hadoop and bulk-loaded into database.
Row-Based Data File Format	SequenceFile [23]	1. This file format is compact in comparison with Text files. 2. The file format supports optional compression. 3. It supports parallel processing. 4. An enormous number of small-sized files can be stored in a 'container,' provided for the same purpose. 5. One of the biggest advantages of this data file format is the support for block-level compression that allows file compression while allowing file splitting for multiple tasks.	1. This file format is not preferred for tools like Hive. 2. The append functionality of the file format just as good and comparable to other file formats. 3. The file format lacks support for multiple languages.	If intermediate data, which is generated between jobs, needs to be saved, then SequenceFile data file format is used.
	Avro [16]	1. The size of serialized data is smallest. 2. It offers block-level compression, allowing file splitting at the same time. 3. This file format maintains the structure of the object. 4. Even if the schema has changed, Avro allows reading of old data. 5. Schema definitions are written in JSON. Therefore, development is considerably simplified in programming languages that possess JSON libraries.	1. Reading and writing processes need schema definition.	Avro is considered best for cases where schema evolution is a key requirement. If the schema is expected to change over time, then Avro is preferred. In fact, Avro is preferred for all Hadoop-based use cases.
Column-Based Data File Format	RCFile [27]	RCFile offer typical benefits associated with columnar databases, which include – 1. It offers good compression. 2. The query performance is better than that for row-oriented databases.	1. There is no support for schema evolution. 2. The write process is slower.	If the use case involves tables that possess many columns and the application requires frequent use of specific columns, then RCFile is the preferred data format.
	ORCFile [13]	1. The compression capabilities of ORCFile are better than that of RCFile. 2. Query processing is also improved when compared to RCFile.	1. There is no support for schema evolution. 2. ORCFile format is not well supported by Apache Impala.	ORCFile and Parquet are used in scenarios where query performance is crucial. However, it has been found that Parquet when used with SparkQL show best improvements in query performance.
	Parquet [14]	1. As is the case with ORCFile format, compression and query performance are good. Moreover, in cases where specific columns are being queried, this data file format is particularly effective. 2. The read performance is good. 3. Schema evolution, in this case, is better than ORCFile format as columns can be added at the end. 4. Storage optimizations are remarkable and it offers file-level as well as block-level compression.	1. Writes are computationally intensive. 2. If the application requires rows of data, then like all columnar databases, Parquet may not be performance-effective in view of the network activity overheads involved.	
	CarbonData [30]	1. It supports update and delete operations, which is crucial for many workflows. 2. CarbonData offers optimizations like bucketing and multi-layer indexing. Therefore, joining two files is more performance-effective and queries are speedier than ever.	1. CarbonData does not support ACID. 2. The size of compressed files is larger than those obtained with ORC and Parquet. 3. CarbonData is a relatively new data file format with many technologies like Athena [33] and Presto [34] not supporting it yet.	CarbonData can be used for variable analytical workloads with typical use cases involving interactive or detailed queries and queries involving real-time ingestion and aggregating/OLAP BI.

In-Memory Data File Format	Arrow [17]	1. This format enables systems to handle big datasets. 2. Same memory can be shared by multiple applications by means of a common data access layer. 3. This file format is optimized for parallel processing and data locality. Moreover, it has been developed for Single Instruction Multiple Data (SIMD). 4. It allows processing of scan as well as random access workloads. 5. The overheads associated with streaming messages and RPC are significantly reduced. 6. Apache Arrow can be used with GPUs. 7. The columnar format provided by Apache Arrow is 'fully shredded' and supports nested and flat schemas.	1. As of now, predicate push down implementation is left to the engine. Although, Apache Arrow is expected to reusable, fast vectorized operations, but efforts in this direction are yet to take shape.	Apache Arrow is best suited for vectorized processing that involve Single Instruction on Multiple Data (SIMD).
Data Storage Services	Kudu [15]	1. OLAP workloads can be processed quickly with Kudu. 2. Kudu can be seamlessly integrated with Spark, Hadoop and its ecosystem. 3. It can be tightly integrated with Apache Impala, which is an effective alternative to Parquet with HDFS. 4. The system is highly available. 5. The data model provided is structured. 6. The management and administration of Kudu is simple. 7. The consistency model is flexible and strong. 8. Random and sequential workloads can be simultaneously executed with high performance.	1. There are no data restore or backup options inbuilt in Kudu. 2. There are some security limitations like authorization available only at the system level and no inbuilt support for data encryption. 3. Kudu does not support automatic partitioning and repartitioning of data. 4. There are other schema-level limitations like lack of support for secondary indexes and multi-row transactions.	Kudu has been created for applications centered on time series data and for applications like online reporting and machine data analytics.

In case of row-oriented file formats, contiguous storage of rows is done in the file. On the other hand, in case of column-oriented formats, rows are split and values for a row-split are stored column–wise. For example, the values for the first column of a row split are stored first and so on. Therefore, columns not required for a query's processing can be omitted during data access. In other words, row-oriented formats are best suited for cases in which fewer rows are to be read, but many columns for the row are required. On the other hand, if a small number of columns are required, then column-oriented file formats fit better.

It is important to note that column-oriented data file formats require a buffer for row splitting as it works with more than one row at a time. Moreover, it is not possible to control the writing process. If the process fails, it is not possible to recover the current file. This is the reason why column-oriented formats are not used for streaming writes. On the other hand, the use of Avro allows read up to the point until which sync had occurred. Flume makes use of row-oriented formats because of this property [35].

Systems are developed in such a manner that seeks to a disk are kept to a bare minimum and thus, latency is optimally reduced. This is an efficient storage mechanism for transactional workloads for which data is written row-wise. For analytical workloads, a large number of rows needs to be accessed at the same time. However, a subset of columns may have to be read for processing a query. In such scenarios, row-oriented format is inefficient considering the fact that all the columns of multiple rows will have to be read even if all these columns are not required. The use of column-oriented format is expected to reduce the number of seeks, improving query performance. Although, writes are slower in this case, but for analytical workloads, this is expected to work well as the number of reads are expected to outnumber writes.

RCFile [27] is the most primitive column-based data file format and ORCFile is an optimized version of the same. Although, ORCFile [13] is considered apt for applications with ACID transactions and offers fast access with its indexing feature, Parquet is promoted by Cloudera [36] and is known to perform best with Spark [8]. The most advanced and recent file format in this category is CarbonData [30], but owing to its newer status, its compatibility with technologies is questionable. This makes Parquet the most popular column-based data file format. Arrow and Kudu [15] support columnar representation with the difference that Arrow supports 'in-memory' storage while Kudu provisions storage that is mutable on disk as against Parquet format, which is immutable on disk.

The tradeoffs for using immutable data file format (Parquet [14]) include higher throughput for write, easier concurrent sharing, access and replication, and no overheads related to operations. However, for making any modifications, dataset needs to be rewritten. Mutable solutions (Kudu [15]) allows higher flexibility in the compromise between speeds for making updates and reads. The latency for short access is lower owing to quorum replication and primary key indexing. Moreover, the semantics are similar to that of databases. With that said, a separate daemon is required for management.

When comparing Parquet's on-disk storage with transient in-memory storage of Arrow [17], it is important to note that the former allows multiple query access with priority given to I/O reduction, but CPU throughput must be good. However, on-disk storage is deemed appropriate for streaming access only. On the other hand, in-memory storage supports streaming, as well as random access, and is focused towards execution of a single query. In this case, CPU throughput is prioritized even though I/O throughput must also be good. Some projects make use of Arrow and Parquet together. Pandas [37] is an example of such a usage. The data frame from Pandas can be saved onto Parquet, which can then be read onto Arrow. The ability of Pandas to work on columns of Arrow, allows it to easily integrate with Spark.

V. CONCLUSION

This importance of big data file formats can be understood from the fact that different technologies in a heterogeneous technological environment need to share data to operate. The most optimized method for such data sharing is the use of common data file formats. This research paper classifies available big data file formats into five categories namely text-based, row-based, column-based, in-memory and data storage services.

Text-based formats have lost utility in the era of big data. However, JSON and XML format usage is common considering the fact that they are lightweight and human readable. Hadoop provides better ways to store and format data on files. Closest to the relational way of storing data is row-based format. SequenceFile and Avro belong to this category, of which former is the most primitive row-based format provided to users.

Other specialized row-based formats that use SequenceFile at their base are also available in Hadoop. These include MapFile, SetFile and BloomMapFile. Avro is the most commonly used row-based format and is preferred for applications that may require all or a majority of the columns.

In most big data scenarios, column-based file formats are known to perform better than their row-based counterparts, as most queries require retrieval of a few columns for multiple rows. Three types of column-based formats are available namely mutable, on-disk storage (Kudu), immutable, on-disk storage (Parquet) and in-memory storage (Arrow). It is important to mention that even though RCFile and ORCFile are also available, Parquet is the most popular column-based format for that category.

Some systems use a combination of these formats depending on the requirements of the application. Future work in this area includes quantitative analysis of available file formats using industry use cases. Besides this, the selection criteria for big data file formats can be diversified, by analyzing the performance file formats for big data technologies such as Spark.

ACKNOWLEDGMENT

This work was supported by a grant from “Young Faculty Research Fellowship” under Visvesvaraya PhD Scheme for Electronics and IT, Department of Electronics & Information Technology (DeitY), Ministry of Communications & IT, Government of India.

REFERENCES

1. M. Hausenblas and J. Nadeau, “Apache drill: interactive ad-hoc analysis at scale,” *Big Data*, vol. 1, no. 2, pp. 100–104, 2013.
2. L. George, *HBase: the definitive guide: random access to your planet-size data*. O’Reilly Media, Inc., 2011.
3. F. Chang *et al.*, “Bigtable: A distributed storage system for structured data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.
4. S. Ahmed, M. U. Ali, J. Ferzund, M. A. Sarwar, A. Rehman, and A. Mehmood, “Modern data formats for big bioinformatics data analytics,” 2017.
5. T. White, *Hadoop: The definitive guide*. O’Reilly Media, Inc., 2012.
6. R. K. Mishra, “The Era of Big Data, Hadoop, and Other Big Data Processing Frameworks,” in *PySpark Recipes*, Berkeley, CA.: Apress, 2018, pp. 1–14.
7. K. Bansal, P. Chawla, and P. Kurle, “Analyzing Performance of Apache Pig and Apache Hive with Hadoop,” in *Engineering Vibration, Communication and Information Processing*, Springer, Singapore, 2019, pp. 41–51.
8. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” *HotCloud*, vol. 10, no. 10, p. 95, 2010.
9. E. Asemota, S. Gallagher, G. Mcrobbie, and S. Cochran, “Defining case based reasoning cases with xml.,” 2018.
10. J. J. C. Gallego and M. D. M. Fernández-Bermejo, “Analysis of the impact of file formats for open data analytics efficiency: a case study with R,” *GSTF J. Comput.*, vol. 5, no. 1, 2018.
11. J. L. Rebelo Moreira, L. Ferreira Pires, and M. Van Sinderen, “Semantic Interoperability for the IoT: Analysis of JSON for Linked Data,” *Enterp. Interoperability Smart Serv. Bus. Impact Enterp. Interoperability*, pp. 163–169, 2018.
12. P. Chamoso, A. González-Briones, S. Rodríguez, and J. M. Corchado, “Tendencies of technologies and platforms in smart cities: A state-of-the-art review,” *Wirel. Commun. Mob. Comput.*, 2018.
13. A. Gupta, M. Saxena, and R. Gill, “Performance Analysis of RDBMS and Hadoop Components with Their File Formats for the Development of Recommender Systems,” in *2018 3rd International Conference for Convergence in Technology (I2CT)*, 2018, pp. 1–6.
14. G. Gershinsky, “Efficient Analytics on Encrypted Data,” in *11th ACM International Systems and Storage Conference*, 2018, p. 121.
15. B. Quinto, *Next-Generation Big Data: A Practical Guide to Apache Kudu, Impala, and Spark*. Apress, 2018.
16. G. S. Hukill and C. Hudson, “Avro: Overview and Implications for Metadata Processing,” 2018.
17. R. M. White, “Open Data Standards for Administrative Data Processing.,” 2018.
18. T. Sharma, V. Shokeen, and S. Mathur, “Comparison of Approaches of Distributed Satellite Image Edge Detection on Hadoop,” in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 645–649.
19. K. Sansanwal, G. Shrivastava, R. Anand, and K. Sharma, “Big Data Analysis and Compression for Indoor Air Quality,” *Handb. IoT Big Data*, vol. 1, 2019.
20. M. Rodrigues, M. Y. Santos, and J. Bernardino, “Experimental Evaluation of Big Data Analytical Tools,” in *European, Mediterranean, and Middle Eastern Conference on Information Systems*, Springer, Cham., 2018, pp. 121–127.
21. J. Gilchrist, “Parallel data compression with bzip2,” *Proc. 16th IASTED Int. Conf. parallel Distrib. Comput. Syst.*, vol. 16, pp. 559–564, 2004.
22. P. A. Carter, “Understanding JSON,” in *SQL Server Advanced Data Types*, Berkeley, CA: Apress, 2018, pp. 181–200.
23. M. A. Ahad and R. Biswas, “Handling Small Size Files in Hadoop: Challenges, Opportunities, and Review,” in *Soft Computing in Data Analytics*, Singapore: Springer, 2019, pp. 653–663.
24. K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” *2010 IEEE 26th Symp. Mass Storage Syst. Technol. MSST2010*, 2010.
25. Y. Xue, Y. A. Tan, C. Liang, C. Zhang, and J. Zheng, “An optimized data hiding scheme for deflate codes,” *Soft Comput.*, vol. 22, no. 13, pp. 4445–4455, 2018.
26. J. R. Carroll and F. R. Robertson, “A Comparison of Phasor Communications Protocols (No. PNNL-28499),” 2019.
27. S. K. Makki and M. R. Hasan, “Measuring the Performance of Data Placement Structures for MapReduce-based Data Warehousing Systems,” *Int. J. New Comput. Archit. Their Appl.*, vol. 8, no. 1, pp. 11–21, 2018.
28. M. Rhu, M. O’Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, “Compressing DMA engine: Leveraging activation sparsity for training deep neural networks,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 78–91.
29. M. Kösters *et al.*, “pymzML v2.0: introducing a highly compressed and seekable gzip format,” *Bioinformatics*, vol. 34, no. 14, pp. 2513–2514, 2018.
30. H. Jiang and A. J. Elmore, “Boosting data filtering on columnar encoding with SIMD,” in *Proceedings of the 14th International Workshop on Data Management on New Hardware*, 2018, p. 6.
31. W. Liu, F. Mei, C. Wang, M. O’Neill, and E. E. Swartzlander, “Data compression device based on modified LZ4 algorithm,” *IEEE Trans. Consum. Electron.*, vol. 64, no. 1, pp. 110–117, 2018.

32. 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.
33. P. Love and T. G. Hartland, "Using AWS Athena analytics to monitor pilot job health on WLCG compute sites," ATL-SOFT-PROC-2018-059, 2018.
34. P. Pant, P. Kumar, I. Alam, and S. Rawat, "Analytical Planning and Implementation of Big Data Technology Working at Enterprise Level," in *Information Systems Design and Intelligent Applications*, Singapore: Springer, 2018, pp. 1031–1043.
35. I. Kovačević and I. Mekterovic, "Novel BI data architectures," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1191–1196.
36. R. Ahuja, "Hadoop Framework for Handling Big Data Needs," in *Handbook of Research on Big Data Storage and Visualization Techniques*, IGI Global, 2018, pp. 101–122.
37. J. Peltenburg, J. van Straten, M. Brobbel, H. P. Hofstee, and Z. Al-Ars, "Supporting Columnar In-memory Formats on FPGA: The Hardware Design of Fletcher for Apache Arrow," in *International Symposium on Applied Reconfigurable Computing*, 2019.

AUTHORS' PROFILE

Samiya Khan is a Ph.D. Student in the Department of Computer Science, Jamia Millia Islamia, New Delhi, India since October 2015. Samiya received B.Sc. degree in Electronics and M.Sc. degree in Informatics from University of Delhi, India. She is currently working on a Cloud-based Framework for Big Data Analytics and has several research papers related to the field in reputed publications. Her research interests also include data-intensive computing, big data, machine learning, deep learning and natural language processing.

Mansaf Alam received his doctoral degree in Computer Science from Jamia Millia Islamia New Delhi in the year 2009. He is currently working as an Associate Professor in the Department of Computer Science, Jamia Millia Islamia. He is also the Editor-in-Chief, Journal of Applied Information Science. He is an Editorial Board of some reputed International Journals in Computer Sciences and has published about 24 research papers. He also has two books entitled "Concepts of Multimedia, Book" and "Digital Logic Design, PHI" to his credit. His areas of research include Cloud Computing, Big data analytics, Object-Oriented Database System (OODBMS), Genetic Programming, Bioinformatics, Image Processing, Information Retrieval and Data Mining.