Indexing in Big Data Mining and Analytics

Ali Usman Abdullahi, Rohiza Ahmad, and Nordin M. Zakaria

Abstract Big data analytics is one of the best ways of extracting values and benefits from the hugely accumulated data. The rate at which the global data is accumulating and the rapid and continuous interconnecting of people and devices is overwhelming. This further poses additional challenge to finding even faster techniques of analyzing and mining the big data despite the emergence of specific big data tools. Indexing and indexing data structures have played an important role in providing faster and improved ways of achieving data processing, mining and retrieval in relational database management systems. In doing so, index has aided in data mining by taking less time to process and retrieve data. The indexing techniques and data structures have the potential of bringing the same benefits to big data analytics if properly integrated into the big data analytical platforms. A lot of researches have been conducted in that direction, and this paper attempts to bring forward how the indexing techniques have been used to benefit the big data mining and analytics. Hence, this can bring the impact that indexing has on RDBMS to the folds of big data mining and analytics.

Keywords Big data · Big data analytics · Data mining · Indexing techniques · Index

A. U. Abdullahi (✉)
Computer Science Education Department, Federal College of Education (Tech), Gombe, Nigeria
e-mail: usmanali@fcetgombe.edu.ng
URL: http://www.fcetgombe.edu.ng

R. Ahmad · N. M. Zakaria
Computer and Information Sciences Department, Universiti Teknologi PETRONAS, Seri Iskandar, Perak, Malaysia
e-mail: rohizaahmad@utp.edu.my

N. M. Zakaria
e-mail: nordinzakaria@utp.edu.my

H. Chiroma et al. (eds.), *Machine Learning and Data Mining for Emerging Trend in Cyber Dynamics*, https://doi.org/10.1007/978-3-030-66288-2_5

1 Introduction

The exponential growing nature of global data was collated and presented by the Statista in their report titled—Volume of data/information created worldwide from 2010 to 2025. The report indicated that the world's overall volume of both created and copied data as of that 2020 year would be 50.5ZB. Also, this volume is expected to rise three-fold within five years and is estimated to be 175 by 2025 [38]. This fact gives a sense of hugeness of data size that is termed as big data [9, 10]. This global data is accumulated from various forms, comprising structured, semi-structured and unstructured masses of data that need analysis so dearly. Since the data are collated and stored into datasets, then those enormous datasets are also referred to as big data/datasets [5, 34, 44].

According to Chen et al. [9, 10], the increase in volume indicates how big the generation, the collection and the scaling of data masses have become. While increase in velocity indicates the need for timely and rapid collection and analysis of big data in order to maximally utilize its commercial value, the increase in variety indicates that big data comprises various forms which may include unstructured and semi-structured, besides the usual structured data [4].

The IDC on its part presented a different opinion on big data. In its 2011 report, IDC viewed big data from its technological and architectural angle as the data is designed for extraction of economic value. The economic value comes from the very large volume and widely varied data, which have high velocity of discovery, capture and analysis [44].

The IDC definition added another 'V' to make it 4Vs model, with the fourth 'V' being value. Therefore, a broader definition of the big data could be derived from all earlier ones as the term is used in describing enormous datasets, whose contents are characterized by the 4Vs: (i) Volume—very large amount of data; (ii) Variety—different forms of the data, i.e., structured, semi-structured and unstructured gathered from different sources including images, documents and complex record; (iii) Velocity—the data has been constantly changing contents, which come from complementary data collection, archived and streamed data [6, 10, 45]; and (iv) Value—very huge value and very low density [10].

In addition, recent literatures include veracity as the fifth 'V', the characteristic of the big data after being convinced that none of the earlier described characteristics of big data have covered that. By veracity, it means that there are uncertainty and effect of accuracy on the quality of collected data [4, 5, 42, 44].

The value in big data is extracted by proper and efficient retrieval of the data from the big datasets. Thus, fast processing of the retrieved data is determinant to faster and timely analysis and access to the required data. Indexing is one of the most useful techniques for faster data retrieval during processing and accessing. Therefore, the industrial 4.0 data-driven processes are in need of such faster data retrieval. Once the retrieval and access processes involving big data usage are made faster, a lot of benefits are achieved. These benefits include energy/power saving and improving hardware durability and reduce heat generation.

1.1 Objective of the Chapter

The main objective of this chapter was to present indexing techniques that can be used in searching and effective retrieval of data during data mining. The strategy of all the indexing techniques is to restrict the amount of input data to be processed during the mining of any dataset. The chapter highlights those indexing techniques that have the potentials of working better with big data.

1.2 Taxonomy of the Chapter

The focus of the chapter is to include in it all relevant literatures that are searched and downloaded from the major publication databases. The databases include IEEEXplore, DBLP, Scopus, Springer and others. Then, titles, abstracts, introductions and conclusions of papers that covered application of indexing in big data mining and analytics were described. This was done using selection criteria in order to pick the papers that matched and have highlighted the structure of various indexing approaches used for big data mining and analytics. In addition, the chapter identifies the indexing approaches that have potential for big data mining and those that have less potential. Figure 1 depicts the diagrammatic sketch of the taxonomy used for the chapter.

The chapter's perspective is to center the discussion of literatures on the indexing approaches, their data structure, their mode of application, their pros and cons, and prospect for big data mining. In addition, the chapter attempts to cover indexing from its application on RDBMS up to it's use in big data mining and analytics. This may enable specialist and practitioners of big data mining and analytics to have wider view on the indexing approaches, and when, how and where to apply each of the approaches for better results.

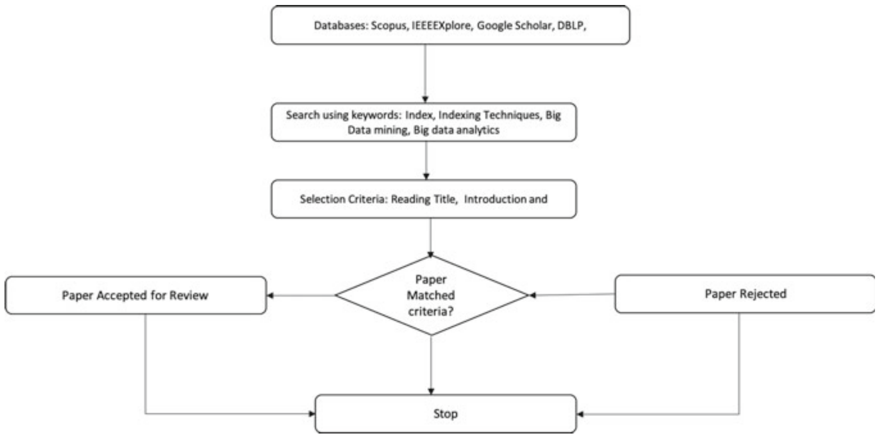

Databases: Scopus, IEEEXplore, Google Scholar, DBLP,
Search using keywords: Index, Indexing Techniques, Big Data mining, Big data analytics
Selection Criteria: Reading Title, Introduction and
Paper Matched criteria?
Paper Accepted for Review
Paper Rejected
Stop

Fig. 1 A sketch of the taxonomy used for the chapter

The remaining part of the chapter is organized as follows. Sect. 2 presents the basic types of the indexing, Sect. 3 discusses the online indexing which is improvement of the basic indexing. Section 4 enumerates the indexes inbuilt to MapReduce process, which is a dedicated process of big data processing. Then, Sect. 5 presents the user-defined indexes, and finally, Sect. 6 concludes the paper.

2 Index and Indexing

Indexing, as information retrieval technique, is the process of generating all the suitable data structures that allow for efficient retrieval of stored information [35]. The term index refers to the suitable data structure needed, to allow for the efficient information retrieval [26]. Usually, the data structures used for indexing in most cases do not store the information itself. Rather, it uses other data structures and pointers to locate where the data is actually being stored. A meta index is also used to store additional information about stored data like the external name of a document, its length, which can be ranked as the output of the index.

In RDBMS, the index is a data structure that speed up the operation of data retrieval from database tables. The index comes with additional cost for writing/reading it into/from the index table. This leads to more storage usage in order for the extra copy of data created by the index to be maintained. The indexes are used in locating data quickly by going straight to the data location without having to search all the rows in the database table every time that table is accessed. Indexes are the bases for providing rapid random lookups and efficient access of ordered recorded.

While in big data, if index is to be implemented the same way as it works with the RDBMS, there is no doubt that it will be so big to the extent that the intended fast record retrieval may not be achieved. Therefore, there is a need for a closer look at the indexing technique in a way that will be reasonably small and maintain the targeted goal of faster record retrieval.

2.1 Index Architecture and Indexing Types

The index has two architectures: non-clustered and clustered index, as shown in Fig. 3. The non-clustered architecture presents data in an arbitrary order, but maintains a logical ordering in which rows may be spread out in a file/table without considering the indexed column expression. This architecture uses a tree-based indexing that has a sorted index keys and pointers to record at its leaf node level. The non-clustered index architecture is characterized by having the order of the physical rows of the indexed data differing with the order in the index [32]. The non-clustered index sketch is displayed in Fig. 3 (Fig. 2).

On the other hand, clustered index architecture changes the blocks of data into a certain distinct order to match the index. This results in the ordering of the row

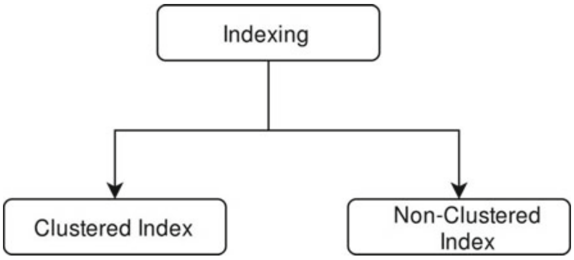


Fig. 2 A sketch of index types

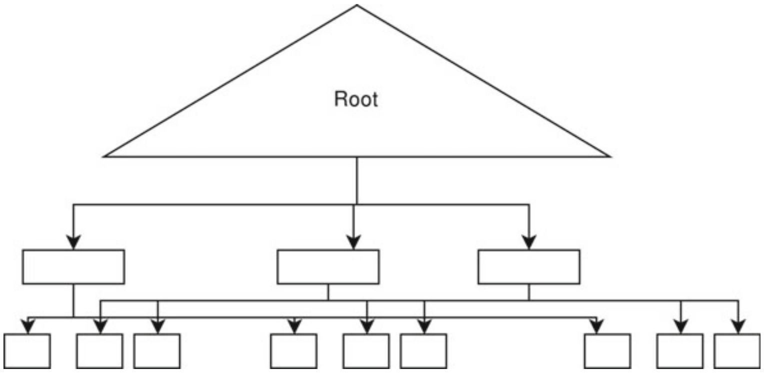


Fig. 3 A sketch of non-clustered index

data. It can greatly increase the overall speed of retrieval in a sequential accessed data or reverse order of the index or among a selected range of items. The major characteristic of the clustered index architecture is that the ordering of the physical data rows is in accordance with that of the index blocks that points to them [29]. The clustered index sketch is displayed in Fig. 4.

There are many types of indexing in existence, and below is the overview of some of them, as well as some of the studies involved each of them and their applicability in big data situation.

2.2 *Bitmap Index*

The bitmap index uses bit array called bitmaps to store the bulk of its data. Bitmap is a special type of index that uses bitwise logical operation on the bitmaps to answer most of the queries run against it. The bitmap index works basically in situations where index values are repeated very frequently unlike other index types commonly used. The other types are most efficient when indexed values are not repeated at all or they are repeated smaller number of times. The gender field of a database table is

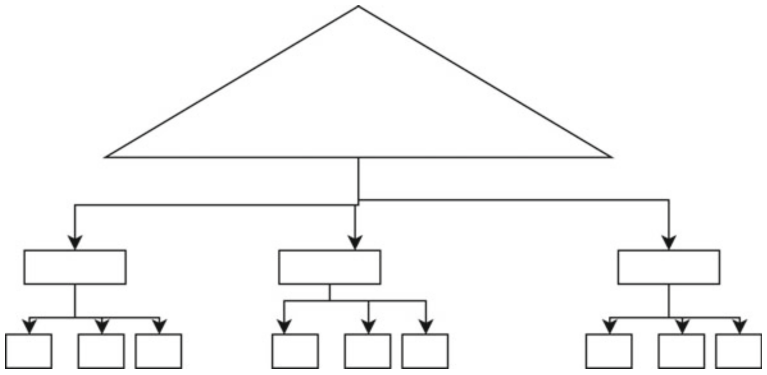
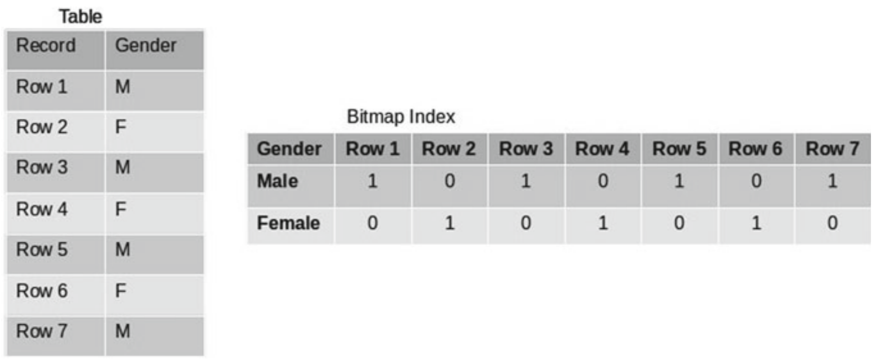


Fig. 4 A sketch of clustered index

a good example for a bitmap index. This is so because no matter how many tuples there are in a database table, the field will only have two possible values: male or female. A typical bitmap index data structure is shown in Fig. 5.

The bitmap index has been used in wide variety of areas and in big data analytics. For instance, bitmap index application was used in the aspect of index compression. The approach helps the compression to work better with high cardinality attribute data. Wu et al. [39] present a study and analysis of some of the compression techniques that use bitmap indexing, namely, byte-aligned bitmap compression (BBC) and word-aligned hybrid (WAH). These techniques were able to reduce compressed data sizes and improve their performance. The authors' motivation was the fact that most of the empirical researches do not include comparative analysis among these different techniques, and the result of their own work showed that compressed bitmap indexes appeared to be smaller in size compared to that of B+-Tree, with WAH occupying half of the space of B+-Tree while the BBC occupies half the space of WAH.



Table

Record	Gender
Row 1	M
Row 2	F
Row 3	M
Row 4	F
Row 5	M
Row 6	F
Row 7	M

Bitmap Index

Gender	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
Male	1	0	1	0	1	0	1
Female	0	1	0	1	0	1	0

Fig. 5 A bitmap index form from gender table

Another work was done by Fusco et al. [13] using bitmap index as a compression approach to minimize CPU workload and consumption rate of disk. The platform used for the work was a streaming network data, which requires a real-time indexing. The authors introduced what they called COMPAX, a variant of compressed bitmap index that supersedes the word-aligned hybrid (WAH) in terms of throughput of indexing, shorter retrieval time and higher compression rate. The NETwork Flow Index (NET-FLI), which highly optimizes real-time indexing as well as data retrieval from larger-scale repositories of network, was used. The NET-FLI synergies COMPAX and locality-sensitive hashing (LSH) are used for streaming reordering in an online setup to achieve the target of the research. This combination results in higher insertion rates of up to 1 million flows per second many folds over what is obtainable in typical commercial network flow. The ISPB also allows the performance of complex analysis jobs by administrators.

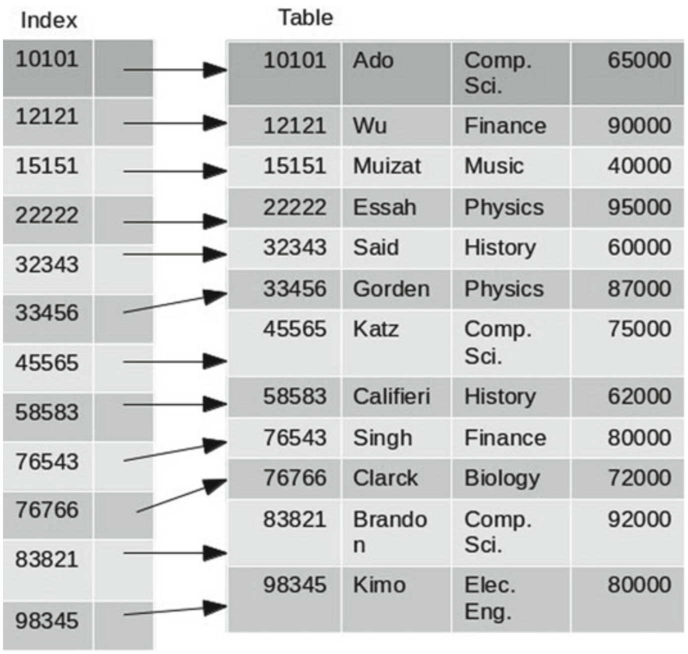
In addition, an effort was made to automate indexing process as well as resolving the index selection problem (ISP) using bitmap by [27]. They came up with a technique that merges the features of index selection techniques (IST) and those of linear programming for optimization to minimize cost. The result was a new method that solves ISP externally and uses optimizer for choosing the set of indexers to be used. The clustering data mining technique was used and when benchmarked it outperformed Microsoft SQL Server Index Selection Tool (IST) in terms of speed of selection and suggestion of indexes. Even though the bitmap index was designed for RDBMS, it has also been used on some HLQLs on big data to improve information retrieval. For example, a group of students incorporated bitmap index into Hive to improve the retrieval of Facebook users information using gender field.

The bitmap technique as a candidate of indexing in big data has been tried, and in some specific situation it gives some improved results. However, the bitmap index does generate large volume of data as its data structure alongside the volume of the big data itself. Hence, it cannot be used for general data retrieval in big data because big data contains variety of data values.

2.3 Dense Index

The dense indexing approach uses a file/table with pair of keys and pointers to each record in the data file/table, which is sorted. Every key in the dense index is associated to a specific pointer to one of such records. If the underlining architecture of the indexes is a clustered one with duplicate keys, dense index just points to the first record with the said key [15]. Figure 6 displayed a sample of the dense index. In the dense index, each index entry consists of a search key and a pointer. The search key holds the value to be searched while the pointer stored the identifier to the disk location containing the corresponding record and the offset that identifies the point where the record starts within the block.

The dense index as an ordered index either stores index entry for all records when the approach it is using is non-clustered, or it just stores the index entry to the first



Index
10101
12121
15151
22222
32343
33456
45565
58583
76543
76766
83821
98345

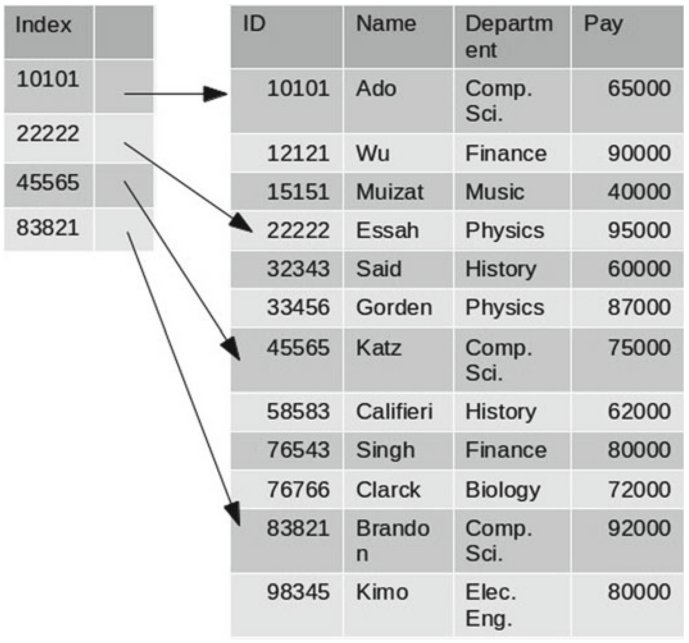
10101	Ado	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Muizat	Music	40000
22222	Essah	Physics	95000
32343	Said	History	60000
33456	Gorden	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Clarck	Biology	72000
83821	Brando n	Comp. Sci.	92000
98345	Kimo	Elec. Eng.	80000

Fig. 6 A dense index on employee table

search key when using the clustered index approach [35]. The dense index uses file in storing its index data structure and there is a dedicated pointer to each record and the data has to be ordered. These two facts suggested that the index is going to grow so big that it will be close to the size to that of the stored data. Hence, there are no studies using dense index in big data retrieval.

2.4 Sparse Index

The sparse indexing method also uses a file/table that contains pair of search keys and pointers. The pointers are pointing to blocks instead of individual records in the data file/table, which is sorted in the order of the search keys. In sparse index, index entries appeared for some of search key values. An index entry is associated to a specific pointer to one of such blocks. If the underlining architecture of the indexes is a clustered one with duplicate key, then the index just points to the lowest search key in each block [15]. To locate any search key's record, an index entry is searched with the largest value that is less than or equal to the given search key value. Then, the searching starts at the record pointed to by the index entry and moves down until the target is found [32, 35]. Figure 7 depicts the explanation given above for sparse index.



Index	
10101	
22222	
45565	
83821	

ID	Name	Department	Pay
10101	Ado	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Muizat	Music	40000
22222	Essah	Physics	95000
32343	Said	History	60000
33456	Gorden	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Clarck	Biology	72000
83821	Brandon	Comp. Sci.	92000
98345	Kimo	Elec. Eng.	80000

Fig. 7 A sparse index on employee table

The dense and sparse indexes are the most common types of ordered index that most relational databases use for generating query execution plans. However, for both the dense and sparse index types, the use of files/tables to keep the pairs of search keys and pointers may make them very unsuitable for big data indexing due to two reasons: (1) The records and block of data file for big data will be distributed over different clusters, which will make it very difficult to maintain such files. (2) The volume of the big data will make the size of such index files to be unnecessarily very large, which may lead to unreasonable costs of space and maintenance time.

3 Online Indexes

3.1 Online Indexing

Most techniques of indexing are having one drawback or the other. Thus, the big data indexing requires the study of other modified indexing techniques. Some of these modified indexes include the work of Chaudhuri et al. [8], which introduced the online indexing. The online index is an extension to the use of external tuning tools to optimize the physical design of a database through the analysis of representative workload set.

This technique works by monitoring the steps of an actual workload without knowing it upfront, and to create an index automatically when one of the query's execution plans is generated. The index is created at the background of a running workload in one complete go. The online indexing has lessened the work on the side of the database administrator and on the side of the system by avoiding the need of creating the index externally.

However, the online indexing usage has not covered the multi and partial indexes, which are necessary and important approaches for query processing in both OTP and OLAP. The results of comparison between the online indexing and the conventional indexing showed that the online index performed better than the conventional one due to the following: knowledge of workload is not required before creating the index. Since the index is created as side effect of query execution, its entries cover only what is specified in the query's predicates; hence, the reason for better performance. The online index can use any of the basic indexing data structure for its implementation, be it B-Tree or ordered tables-based index [8, 16, 21].

Amir et al. [2] used online indexing to solve the problem of managing unbounded length keys that are found in XLM paths, IP addresses, multi-dimensional points, multi-key data and multi-precision numbers. This type of data is categorized as big data. These types of string-based keys are usually atomic and indivisible; hence requires a customized comparison data structure. Their proposed online indexing happens to work with any data structure with a worst-case complexity of $O(\log n)$, which they reported to be the best based on their experiment. A suffix tree was cited as one of the application area of the proposed online indexing [2].

3.2 Database Cracking

Another effort in the direction of 'on-the-fly' indexing was made by Idreos et al. [21]. The authors introduced a combination of automatic index selection and partial indexes called database cracking. This approach uses the side effect of running current query and future ones to refine the structure of its index. The database cracking substituted the normal scanning of stored data for index creation and query processing, with pivoted partitioning of the data using the predicates from the query.

By doing that, one of the partitions will be containing only the tuples that answers the query. The underlining data structure used by the cracker index could be quick sort generated B-Tree or a hash, whose partitions expected to be beneficial to arriving queries.

In database cracking, no external index or external tuning tools are required, but just the monitoring of queries. The present and arriving queries are used to build and optimize the index. Database cracking proves to be advantageous compared to online indexing, and by extension the ordinary query processing. For the fact that the partitioning has to be carried out in small number for every predicate (fan-out of 2–3), it takes a lot of queries to get the index fully optimized [17, 18, 41].

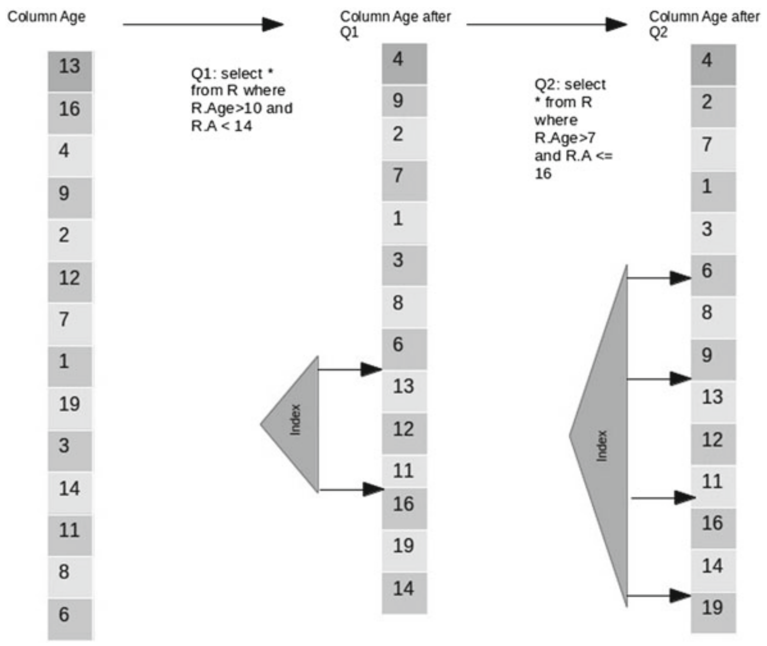


Fig. 8 An example of database cracking

Figure 8 displays an example of database cracking. The database cracking was implemented in one ‘MonetDB’ as a successful alternative to index scanning [7, 28].

However, the database cracking was observed to be CPU-intensive rather than I/O bound. Pirk et al. [28] proposed an enhancement that has taken the database cracking from being CPU bound to I/O bound. Input/output bound operation is best suited for big data processing. The authors used approaches such as predication, vectorization, data parallelism and CPU multi-thread on SIMD instructions to achieve the method. The results of their work showed that it is 25 times faster than the first database cracking.

3.3 Adaptive Merge

Graefe, Goetz, Kuno and Harumi [18] have introduced a modification of database cracking called adaptive merge. The adaptive merge also creates index by using the predicates specified in a query and used the partial index to answer that given query. The process also increments and optimizes the index with the help of the underlining data structure, the partitioned B-Tree. Adaptive merge uses merge sort for the index optimization and works based on ‘monitor queries then build index’ approach.

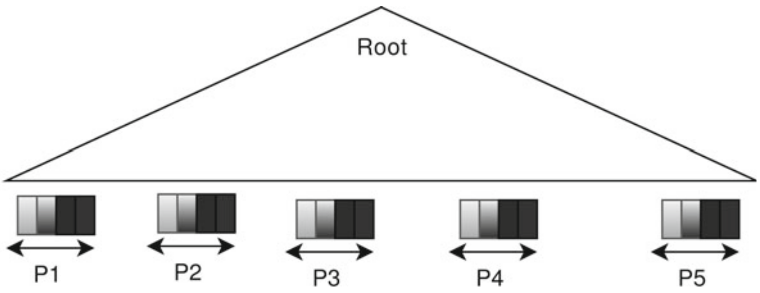


Fig. 9 An adaptive merge index is based of B-Tree

Unlike database cracking that uses partitioning of its data structure and works only on in-memory database, the adaptive merge works on external block access like flash memory and disc as well as in-memory. In terms of performance, the adaptive merge indexing optimizes with far less number of queries compared to database cracking. This is due to the fact that merge process's fan-in is unlimited as against limited fan-out of partitioning [18, 22]. Then Idreos et al. [22] made a further attempt to bring the two approaches into one, by replacing the index initial creation stage of the adaptive merge. The authors suggested a data structure that does not require much resource to build such as array, but optimizes by merging. This attempt was believed to work well with bug datasets. Figure 9 presents the partitioned B-Tree that was generated by adaptive merge index.

The last three indexing methods also referred to as oblivious indexes work by creating the index on-the-fly and automatically. The indexes use the concept of monitoring any issued query and then build the index as the side effect of the query's execution. However, all the above-discussed indexing techniques work with RDBMS and in an OLTP approach. The OLTP usually has short queries that are frequently posed to the database as the main operational system. So, there are number of arriving queries to increment and/or optimize an index if the need arises. On the contrary, the situation is different in the case of OLAP, especially in batch-oriented situations, which are the most common analysis when it comes to big data. Also, the mentioned cases are mostly different when MapReduce is to be used for such analysis.

The drawbacks of the RDBMS in processing big data imply that their corresponding index types have the same drawbacks. This prompted researchers of big data to customize and, in some cases, develop different indexing strategies for the big data analysis as mentioned earlier. These developed indexing strategies for big data information retrieval systems are being used by big data analytics.

3.4 Big Data Analytics Platforms

Upon the realization of the drawbacks of the traditional database systems, research efforts continue with more focus in the direction of fast processing of big data, especially with OLAP queries (big data analytics). The analytical challenge of big data indicates that it has outgrown that capacity of traditional DBMS resources. This further prompts for the development of new technologies to solve the challenge. As a result of the above, Dean and Ghemawat [11] had presented the notable new technologies of Google File System (GFS), which is a distributed file system used for data storage across clusters. The authors also introduced the Google MapReduce programming model, which is used for data analysis and indexing [9, 10, 14, 23].

The MapReduce is a programming paradigm that uses divide and conquer approach to problem-solving. MapReduce uses functional programming approach with only two functions: Map and Reduce. The functions are deployed to the chunks of data that are stored on the distributed file system. The tasks to be carried out by the functions are to be defined by the user and they are executed in parallel on the various blocks of data. The MapReduce works in a key/values filtering and aggregation manner to solve the problem in question [19, 23, 24, 36, 43].

The MapReduce remained the most powerful and the most accepted approach for the big data mining and analytics [24, 37]. One of the major reasons for its popularity among researchers and industry experts is its flexibility. Users can intuitively write code to solve virtually any problem. Besides that, MapReduce also supports very high parallel programming, even though at low level. MapReduce design nature was to eradicate input/output (I/O) problem associated to extract, load and transfer (ELT) [23].

The design migrates the computation to various computing units instead of moving data into memory for computation. Furthermore, the Hadoop MapReduce implementation remains the fastest, one reported, in handling very large data analysis. In addition, larger percentage of research works' experiment involving big data analytics that are found in the literatures are done using the Hadoop MapReduce or they are related to it or its associated tools. Lastly, reports and releases coming from the big IT companies indicate that most of them use tools that are supported by the Hadoop/MapReduce in performing their big data analysis [24, 31, 33, 37]. Table 1 highlights some of the features of three attempted solutions for big data analytics.

4 Inherent Indexes in MapReduce

A simple inverted index is said to be inherently and trivially implemented in MapReduce as one of the effective tasks for textual data retrieval. This is highlighted by Dean and Ghemawat [11] in their original paper on MapReduce and cited by Graef and kuno [18]. When performing inverted index with MapReduce, the map function parses the split covering certain input, and emits sequence of <key, value> pairs.

Table 1 Comparison between big data analytics approaches

S. no.	Big data analytic	Parallel programming approach	Dynamic flexibility	Schema less	SQL support	Extract, load and transfer
1	MapReduce	Yes	Yes	Yes	No	Not directly
2	Algebraic workflow [12]	Yes	No	Yes	Yes	No
3	AterixDB [1]	Yes	No	No SQL Like	Semi-structured	Yes

The reduce function accepts all pairs of the same key, sort the corresponding values and emits <key, list(value)> pair. The set of all output pairs form a simple inverted index. McCreadie [25] deducted that two interpretations of the above scenario can be a per-token indexing or a per-term indexing in relation to the indexing of corpus datasets.

The per-token indexing strategy involves emitting of <term, doc-ID> pairs for each token in a document by the map function. However, the reduce function does the aggregation of each unique term with its corresponding doc-ID to obtain the term frequencies (tf), after which the completed posting list for that term is written to disk. So, if a term appears tf times it will be indicated as such. The advantage of this strategy is that it makes the map phase very simple. However, it has the potential of costing more memory and time due to storage of large intermediate results, network traffic and prolonging sort phase (Fig. 10).

On the other hand, the per-term indexing uses the map function to emit tuple in the form <term, (doc-ID, tf)>, and this reduces the number of emit operation as only unique term per document is emitted. The reduce function in this inter-operation only sorts the instances by document to obtain the final posting list sorted by ascending doc-ID. Ivory information retrieval system uses this approach. Also, a combiner function can be used to generate tfs by performing a localized merge on each map task's output (Fig. 11).

4.1 Per Document Indexing

This is the inverted indexing technique used by Nutch platform on the Hadoop to index document for faster search. Nutch tokenized the document during map phase and the map function emits tuples in the form of <document, doc-ID >, while the reduce phase writes all index structures. Though this strategy emits less, the value of each emit used to have more data and have reduced intermediate results, thus achieving higher levels of compression than single terms. Documents are indexed on same reduce task easily due to the sorting of document names [25].

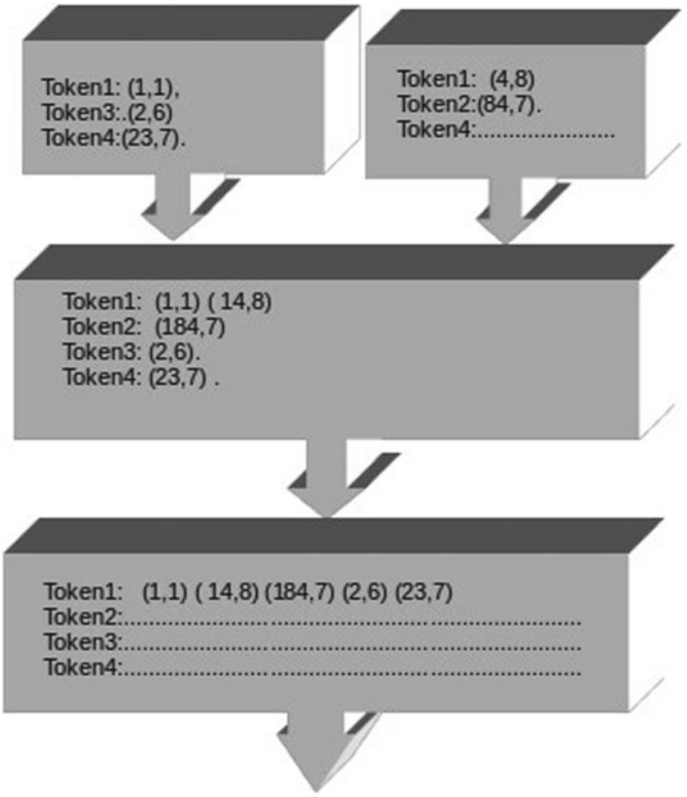

Token1: (1,1),
Token3:.(2,6)
Token4:(23,7).
Token1: (4,8)
Token2:(84,7).
Token4:......................
Token1: (1,1) (14,8)
Token2: (184,7)
Token3: (2,6).
Token4: (23,7) .
Token1: (1,1) (14,8) (184,7) (2,6) (23,7)
Token2:..
Token3:..
Token4:..

Fig. 10 A sample of per token index using MapReduce

4.2 Per-Posting List Indexing

This indexing technique is based on a single-pass indexing, which splits data onto multiple map tasks, with each operating on its own data sub-set. The map task serves as the scanning phase of the single-pass indexing. As this process run on the document, compressed posting lists are built in memory for each term. This partial index is flushed from the map task when the memory run low or when all the documents are processed. The flushing is done by emitting a set in the form of <term, posting list> pairs for all terms present in the memory. Before taking up of intermediate results by reduce task, the flushed partial indexes are sorted and stored on disk first by map number and then by flushed numbers. In order to achieve globally correct ordering of posting list for each term, the posting lists are merged by map number and flushed number. The term, posting lists, is merged together by the reduce

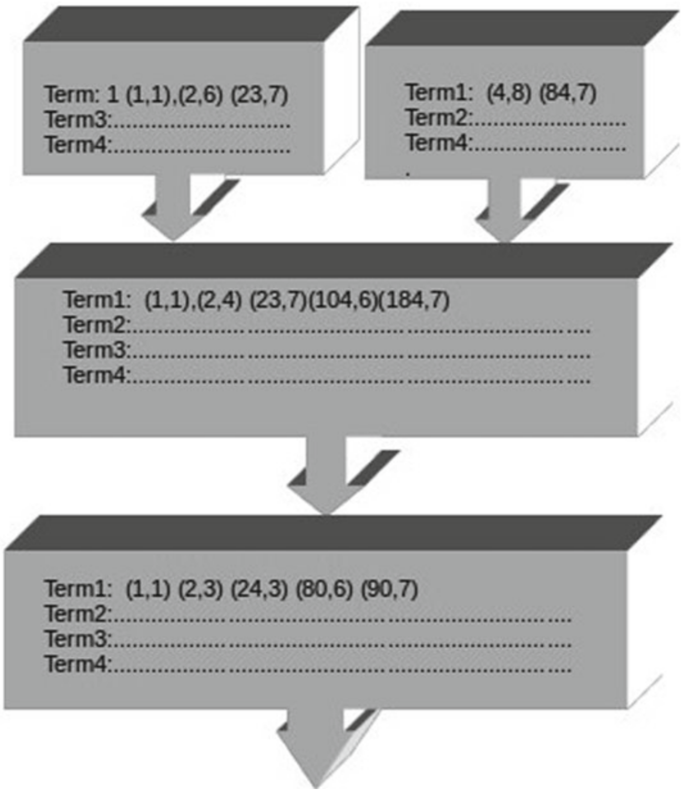

Term: 1 (1,1),(2,6) (23,7)
Term3:..........................
Term4:..........................
Term1: (4,8) (84,7)
Term2:......................
Term4:......................
Term1: (1,1),(2,4) (23,7)(104,6)(184,7)
Term2:..
Term3:..
Term4:..
Term1: (1,1) (2,3) (24,3) (80,6) (90,7)
Term2:..
Term3:..
Term4:..

Fig. 11 A sample of per term index using MapReduce

function to form the standard index comprising full posting lists. The standard index is compressed using EliasGamma technique by storing only the distance doc-IDs [25] (Fig. 12).

All of the four strategies of inverted indexing in MapReduce discussed above are the main task focused on and carried out by the MapReduce job. This is against the primary function of indexing in RDBMS, which is to speed up access of stored data in order to improve the performance of other processes. Thus, the aim of indexing is not only to use but also to improve parallel processing of the document contents. Rather, the primary aim of indexing is to improve the performance of parallel processor itself. This improvement is to be achieved in addition to the underlining parallel processor that MapReduce is programmed to accomplish. Thus, there is a need for additional indexing scheme that works with the parallel processor and serves the same purpose with what index does in RDBMS.

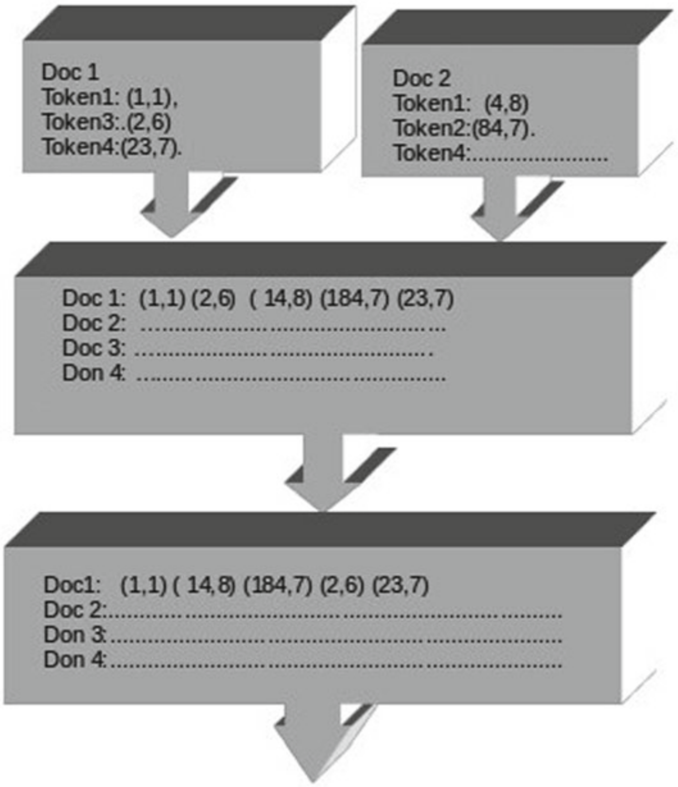

Doc 1
Token1: (1,1),
Token3:.(2,6)
Token4:(23,7).
Doc 2
Token1: (4,8)
Token2:(84,7).
Token4:......................
Doc 1: (1,1) (2,6) (14,8) (184,7) (23,7)
Doc 2: ....
Doc 3: ....
Don 4: ....
Doc1: (1,1) (14,8) (184,7) (2,6) (23,7)
Doc 2:..
Don 3:..
Don 4:..

Fig. 12 A sample of per document index using MapReduce

5 User-Defined Indexing in MapReduce

For the user-defined indexing used in MapReduce and big data analytics, Yang and Parker [40] have employed HDFS's file component as B-Tree nodes to achieve indexing. In their approach each file contains data and pointer to lower files in the tree hierarchy, which is considered its children. During query processing the tree is traversed to locate the required segment of data to be processed using an improved Map-Reduce-Merge-Traverse version of MapReduce. After locating the data, then the map, the reduce and the merge tasks are performed on it to return the record set that answers the given query.

Also, An et al. [3] have used blockIds from the HDFS as the search keys of their B+-Tree-based index. When a given query is to be processed, the B+-Tree-based index is first searched to determine the start and the end of contiguous blocks that formed the index, and the result of the search formed the input data to be scanned. Then, only the blockIds that are returned from such search are used by MapReduce for main query processing. Hence, through this process preventing the full scan of the input data is done. In addition, Richter et al. [30] used the copies of replica stored by HDFS, to index different data attributes, which may likely be used as incoming query's predicates. When a MapReduce query arrives, their library checks the fields contained in the query's predicates and used the clustered index built on that field to return the blockIds of the data required to answer the given query.

Furthermore, in all the mentioned studies, the authors used indexing data structure that scale logarithmically, thereby improving data processing and retrieval. This is done by preventing the MapReduce from full scan of input data by guiding the process to just scanning and processing the data that corresponds to the output of the indexes. Moreover, many other researches were conducted on indexing using different types of big data; however, those researches are closely tied to that type of big data as the indexing data structure and the index implementations are determined by the nature of the data itself [20]. Table 2 displays the summary of the index approaches, their memory requirement and big data potentials.

Table 2 Comparison between big data analytics approaches

Reference	Indexing approach	Data structure	Memory/storage	Potential for big data mining
Wu et al. [39], Fusco et al. [13]	Bitmap	Tabular	Requires large space of memory and storage	Has good potential for big data mining
Gracia et al. [15], and Silberschez et al. [35]	Dense	Tabular	Requires large space of memory and storage	Not a good approach for big data mining
Gracia et al. [15], Rys [32] and Silberschez et al. [35]	Sparse	Tabular	Requires large space of memory and storage	Not a good approach for big data mining
Chaudhuri et al. [8]	Online indexing	Vectors	Requires small space of memory and storage	A good approach for big data mining
Idreos et al. [21]	Database cracking	Arrays	Requires small space of memory and storage	A good approach for big data mining
Graefe et al. [18]	Adaptive merge	Tree-based	Requires average space of memory and storage	A very good approach for big data mining
Yang and Parker [40]	B-Tree	Tree-based	Requires average space of memory and storage	A good approach for big data mining
An et al. [3]	B+-Tree	Tree-based	Requires small space for of memory and storage	A good approach for big data mining
Richter et al. [30]	HAIL	File-based	Requires large space of memory and storage	A good approach for big data mining

6 Conclusion

Moreover, it can be simply deducted from the above reviews that user-defined index and content indexing as tool for optimizing the performance of information retrieval has been successful. It can also be added that there is high potential for improving big data analytics through the use of advanced indexing techniques and data structures. Particularly, if these techniques and data structures are hybridized, improved and customized to work with MapReduce, they will surely improve the performance of big data analytics.

It has been highlighted that MapReduce is one of the most popular tools for big data analytics. However, the low-level nature of its implementations has given rise to the development of HLQLs. The HLQLs ease the programmer's task of handling the analysis. The review also highlighted the different types of index in both RDBMS and those used in big data analytics using different big data analytics platforms, including MapReduce and its index approaches. The improvement can be achieved by changing the indexing approach or using more efficient data structure.

References

1. Alsubaiee, S., Altowim, Y., Altwaijry, H., Behm, A., Borkar, V., Bu, Y., Carey, M., Cetindil, I., Cheelangi, M., Faraaz, K., et al.: Asterixdb: a scalable, open source bdms. Proc. VLDB Endow. **7**(14), 1905–1916 (2014)
2. Amir, A., Franceschini, G., Grossi, R., Kopelowitz, T., Lewenstein, M., Lewenstein, N.: Managing unbounded-length keys in comparison-driven data structures with applications to online indexing. SIAM J. Comput. **43**(4), 1396–1416 (2014)
3. An, M., Wang, Y., Wang, W.: Using index in the mapreduce framework. In: Web Conference (APWEB), 2010 12th International Asia-Pacific, pp. 52–58. IEEE (2010)
4. Bachlechner, D., Leimbach, T.: Big data challenges: Impact, potential responsesand research needs. In: IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech), pp. 257–264. IEEE (2016)
5. Bajaber, F., Elshawi, R., Batarfi, O., Altalhi, A., Barnawi, A., Sakr, S.: Big data 2.0 processing systems: Taxonomy and open challenges. J. Grid Comput. **14**(3), 379–405 (2016)
6. Berman, J.J.: Principles of big data: preparing, sharing, and analyzing complexinformation. Newnes (2013)
7. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/x100: hyper-pipelining query execution. Cidr. **5**, 225–237 (2005)
8. Chaudhuri, S., Narasayya, V.: Self-tuning database systems: a decade of progress. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 3–14. VLDB Endowment (2007)
9. Chen, C.P., Zhang, C.Y.: Data-intensive applications, challenges, techniques andtechnologies: a survey on big data. Inf. Sci. **275**, 314–347 (2014)
10. Chen, M., Mao, S., Liu, Y.: Big data: a survey. Mob. Netw. Appl. **19**(2), 171–209 (2014)
11. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010)
12. Dias, J., Ogasawara, E., de Oliveira, D., Porto, F., Valduriez, P., Mattoso, M.: Algebraic data flows for big data analysis. In: 2013 IEEE International Conference on Big Data, pp. 150–155. IEEE (2013)

13. Fusco, F., Vlachos, M., Stoecklin, M.P.: Real-time creation of bitmap indexes onstreaming network data. VLDB J. Int. J. Very Large Data Bases **21**(3), 287–307 (2012)
14. Gani, A., Siddiqa, A., Shamshirband, S., Hanum, F.: A survey on indexing techniques for big data: taxonomy and performance evaluation. Knowl. Inf. Syst. **46**(2), 241–284 (2016)
15. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database System Implementation, vol. 654, 2nd ed edn. Prentice Hall Upper Saddle River, NJ (2014)
16. Glombiewski, N., Seeger, B., Graefe, G.: Waves of misery after index creation.BTW 2019 (2019)
17. Graefe, G., Idreos, S., Kuno, H., Manegold, S.: Benchmarking Adaptive Indexing, pp. 169–184. Springer (2011)
18. Graefe, G., Kuno, H.: Self-selecting, self-tuning, incrementally optimized indexes. In: Proceedings of the 13th International Conference on Extending Database Technology, pp. 371–381. ACM (2010)
19. Hong, Z., Xiao-Ming, W., Jie, C., Yan-Hong, M., Yi-Rong, G., Min, W.: A optimized model for mapreduce based on hadoop. TELKOMNIKA (Telecommunication Computing Electronics and Control) **14**(4) (2016)
20. Ibrahim, H., Sani, N.F.M., Yaakob, R., et al.: Analyses of indexing techniques onuncertain data with high dimensionality. IEEE Access **8**, 74101–74117 (2020)
21. Idreos, S., Kersten, M.L., Manegold, S.: Database cracking. In: CIDR, vol. 7, pp. 7–10 (2017)
22. Idreos, S., Manegold, S., Kuno, H., Graefe, G.: Merging what's cracked, crackingwhat's merged: adaptive indexing in main-memory column-stores. Proc. VLDB Endow. **4**(9), 586–597 (2011)
23. Khasawneh, T.N., AL-Sahlee, M.H., Safia, A.A.: Sql, newsql, and nosql databases: A comparative survey. In: 2020 11th International Conference on Information and Communication Systems (ICICS), pp. 013–021 (2020)
24. Lee, S., Jo, J.Y., Kim, Y.: Performance improvement of mapreduce process bypromoting deep data locality. In: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 292–301. IEEE (2016)
25. McCreadie, R., Macdonald, C., Ounis, I.: On single-pass indexing with mapreduce.In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. pp. 742–743. ACM (2009)
26. McCreadie, R., Macdonald, C., Ounis, I.: Mapreduce indexing strategies: Studyingscalability and efficiency. Inf. Process. Manage. **48**(5), 873–888 (2012)
27. Nang, J., Park, J.: An efficient indexing structure for content based multimediaretrieval with relevance feedback. In: Proceedings of the 2007 ACM symposium on Applied computing, pp. 517–524. ACM (2007)
28. Pirk, H., Petraki, E., Idreos, S., Manegold, S., Kersten, M.: Database cracking: fancy scan, not poor man's sort! In: Proceedings of the Tenth International Workshop on Data Management on New Hardware, p. 4. ACM (2014)
29. Ramakrishnan, R., Gehrke, J., Gehrke, J.: Database Management Systems, vol. 3. McGraw-Hill New York (2010)
30. Richter, S., Quian´e-Ruiz, J.A., Schuh, S., Dittrich, J.: Towards zero-overhead staticand adaptive indexing in hadoop. VLDB J. **23**(3), 469–494 (2014)
31. Roy, S., Mitra, R.: A survey of data structures and algorithms used in the contextof compression upon biological sequence. Sustain. Humanosphere **16**(1), 1951–1963 (2020)
32. Rys, M.: Xml and relational database management systems: inside microsoft sqlserver 2005. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 958–962. ACM (2005)
33. Sevugan, P., Shankar, K.: Spatial data indexing and query processing in geocloud. J. Test. Eval. **47**(6) (2019)
34. Shireesha, R., Bhutada, S.: A study of tools, techniques, and trends for big dataanalytics. IJACTA **4**(1), 152–158 (2016)
35. Silberschatz, A., Korth, H.F., Sudarshan, S., et al.: Database System Concepts, vol. 4. McGraw-Hill New York (1997)

36. Silva, Y.N., Almeida, I., Queiroz, M.: Sql: From traditional databases to big data. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 413–418. ACM (2016)
37. Sozykin, A., Epanchintsev, T.: Mipr-a framework for distributed image processing using hadoop. In: 2015 9th International Conference on Application of Information and Communication Technologies (AICT), pp. 35–39. IEEE (2015)
38. Statista: Volume of data worldwide from 2010–2025. https://www.statista.com/statistics/871513/worldwide-data-created/ (2020)
39. Wu, K., Otoo, E., Shoshani, A.: On the performance of bitmap indices for highcardinality attributes. In: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. pp. 24–35. VLDB Endowment (2004)
40. Yang, H.C., Parker, D.S.: Traverse: simplified indexing on large map-reduce-mergeclusters. In: International Conference on Database Systems for Advanced Applications. pp. 308–322. Springer (2009)
41. Ydraios, E., et al.: Database cracking: towards auto-tunning database kernels. SIKS (2010))
42. Zakir, J., Seymour, T., Berg, K.: Big data analytics. Issues Inf. Syst. **16**(2), 81–90 (2015)
43. Zhang, Q., He, A., Liu, C., Lo, E.: Closest interval join using mapreduce. In: 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 302–311. IEEE (2016)
44. Zhang, Y., Ren, J., Liu, J., Xu, C., Guo, H., Liu, Y.: A survey on emerging computing paradigms for big data. Chinese J. Electron. **26**(1) (2017)
45. Zikopoulos, P., Eaton, C.: Understanding big data: Analytics for enterprise classhadoop and streaming data. McGraw-Hill Osborne Media (2011)