Experimental Evaluation of Indexing Between HBase and MongoDB

Ade Hodijah[1,*]

[1] *Department of Computer and Informatics Engineering, Bandung State of Polytechnic, Bandung, Indonesia*
[*] *Corresponding author. Email: adehodijah@jtk.polban.ac.id*

ABSTRACT
HBase and MongoDB are non-relational databases where HBase is column-oriented, and MongoDB is document-oriented. MongoDB uses indexing to speed up the searching process, while HBase does not have secondary indexing, a lookup table was created to save an index of searching. This research used a prototype developed in a similar data model design of the HBase and MongoDB database to compare the response time performance in searching operation using keyword frequency criteria in the document management system. The results show that the primary indexes of HBase is faster than secondary indexes of MongoDB. The time difference between HBase and MongoDB is almost a half.

Keywords: *HBase, MongoDB, Indexing, Document management system*

1. INTRODUCTION

There are many studies doing research in column-oriented vs document-oriented [1], [2] with the result shown that document-oriented is faster than column-oriented for read operation. Based on the previous result, the first experiment had been examined for text searching, where MongDB is faster than HBase as well (Table 1).

Table 1. Search processing time on HBase and Mongodb using dummy data

Database	Records	Average (ms)
HBase	5K	326
	50K	307
	100K	317
MongoDB	5K	72
	50K	74
	100K	77

However, in [3] explained that both HBase and MongoDB performed well, but latency of MongoDB was comparatively less than HBase. In this paper, the search processing time will be examined on more complex of data, where data is designed in a document management system. The need for speed of searching a document requires a system that can access and display documents according to certain search criteria by applying to index, called inverted index [4]. An embedded model data of MongoDB was explored to know what collection design type will support for high performance of reads and a memory capacity utilization of HBase block cache [5] was implemented as well. One of the inverted index implementation had been done in [6] by using IndexedHBase extension, while MapReduce functionality in MongoDB had been done for the inverted index implementation [7]. Therefore, there is still another inverted index exploration that compared in this research with focusing on the best data model design in HBase and MongoDB.

2. METHODOLOGY

There are four variables examined: primary indexes, secondary indexes, block size, and block cache in searching for a document management system. The run time of it is compared in HBase and MongoDB. Testing (runtime experimental) data obtained by a simple application to generate Final Assignment documents as good data sources for scientific studies in knowing the fastest response time performance between HBase and MongoDB. The following are details of the research stages, as shown in Figure 1. Firstly, modeling data was made with column-oriented and document-oriented approach by considering the high performance of reads in HBase and MongoDB. Secondly, the experimental was examined in secondary indexes of MongoDB to get the value of the fastest query run time possibility since there is no secondary index in HBase [8]. Thirdly, the primary indexes of HBase and MongoDB were explored to get the value of the fastest query run time between both of them. Finally, the block cache and block size were examined to

find out more what other ways that can affect the high query performance of HBase.

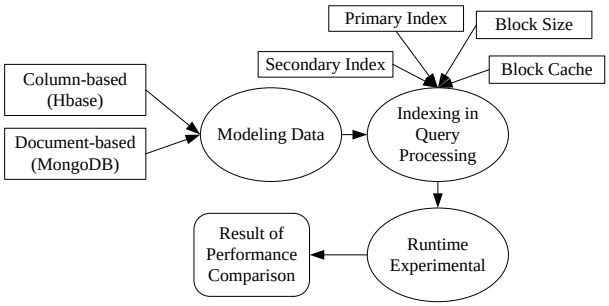


Figure 1 Methodology.

3. RESULT AND DISCUSSION

3.1. Data Model

HBase is column-oriented that uses a table to store data, while MongoDB is document-oriented that uses the collection to store data. The searching operation is performed by using keyword criteria defined in the Abstract part of the document that the search results are sorted from those that are considered most relevant to those that are less relevant based on the counting of appearance in the table (HBase) or collection (MongoDB) that is represented by frequency field. To create indexing in HBase, a developer has to write the code manually to create indexing in HBase because there is no secondary indexing feature built-in [9]. Otherwise, MongoDB indexes are built-in that use a B-tree data structure [10]. Data model design of the document management system in HBase and MongoDB is as follows:

3.1.1. MongoDB

There are three types of collections for searching operation by utilizing the index based on data entered by a user. The first type of collection (BasicInformation) contains all attributes of documents, the second type collection (Document) is used as a lookup table with having a relationship to collection (DocumentFrequent) which is used to sort the DocumentId of Document object based on the frequent value of DocumentFrequent object. Structure of the BasicInformation (Figure 2. a) collection used normalized [11] data model to Document collection, while the data model of DocumentFrequent collection can be implemented by two approaches which are embedded (Figure 2. b) or normalized_1 (Figure 2. c) or normalized_2 (Figure 2.d) to Document collection. The embedded data model is used to avoid joining operations and to improve query performance [12]. Otherwise, on one hand, there is a possibility that the document will increase and have the same name of Word or Title or Author or Year, and the other hand there is a limitation of an embedded document data model in one document, the maximum BSON document size is 16 megabytes [11], so the normalized_1 or normalized_2 is used to give an alternative of the data model. In this research, the embedded data model (Figure 2. b) is used in performance experimental issues based on to make similarity with the HBase data model. In addition, there will be a different mechanism for getting the document between normalized_1 and normalized_2, where the document results in normalized_1 will be joined so that the slices are obtained. Then the document on this slice will be taken. The join process of a document is required because there is a possibility of the same name (Word, Title, Author, Year) appeared with different DocumentFrequent more than one document in that Document collection. Otherwise, there is only one document for the same name in normalized_2, so the join process is performed by collection, which is joined from Document collection to DocumentFrequent collection, while the join process in normalized_1 is performed by the document itself of a Document collection.

```
[
  {
    _id: <ObjectId>
    title: String
    year: Year
    author: String
    keywords: List<String>
    [
      {word: String},
      ...
    ]
  },
  ...
]
```

(a)

```
[
  {
    _id: String
    documentFrequent: List<DocumentFrequent>
    [
      {documentID: <ObjectId>, frequency: Integer},
      ...
    ]
  },
  ...
]
```

(b)

```
[
  {
    _id: String
    documentID: <ObjectId>
    frequency: Integer
  },
  ...
]
```

(c)

```
[
  {
    _id: String
    documentFrequent: List<ObjectId>
    [
      {documentID: <ObjectId>},
      ...
    ]
  },
  ...
]
```

(d)

Figure 2 BasicInformation (a), DocumentFrequent (b), and Document (c) and (d).

The total collection of MongoDB for the document management system is six collections, namely BasicInformation, WordLookup, TitleLookup, AuthorLookup, YearLookup, and DocumentFrequent collection. The documentId field with having String type in Document collections, as seen in Figure 2(b), 2(c) and 2(d) refer to _id field with having ObjectId type (primary key) in BasicInformation collection (Figure 2. a).

The _id field with having String type which is the default MongoDB primary index [10] that is designed in Document collection consists of Word, Title, Author, and Year which are the category of searching operation. In addition, these categories (Word, Title, Author, and Year) is also designed to become the secondary index uses single field index type [10] with replaced the value of _id with having ObjecId type and adding a new field in Document collection that is data with having String type, as seen in Figure 3.

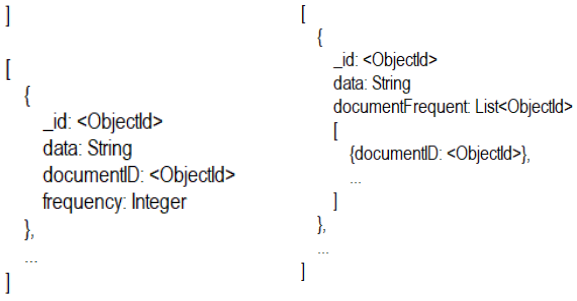


Figure 3 Document collection with data field for secondary index.

The experimental of this research was conducted for the Word category. This is because the focus of this research is to determine the performance of data retrieval by applying the index on HBase and MongoDB. The previous searching mechanism is used to search data based on an input or only the Word category. However, when the user inputs data that is more than one criterion, a joining process is needed from all queries made in each category to get the slices (intersection) that will be retrieved as the document search result.

3.1.2. HBase

There are two types of tables in HBase to perform searching by index. The first table serves as a place to store complete information from the document, and the other table serves as a lookup table to store search index information. The first table consists of the Information column family (CF: Information) as necessary information of documents stored that is Title, Author, and Year and Word column family (CF: Words) as a counter of a keyword on the Abstract part of the Final Assignment document. This model aims to speed up the reading process of getting all data with one read operation without reference content from other tables, as seen in Table 2. Since the number of Word in each DocumentID is unpredictable, the column family of Keywords was designed in a schemaless data model and the column family of DocumentFrequent as well.

Table 2. BasicInformation table in HBase

RowKey	Information			Keywords		
DocumentId	Title	Author	Year	Word_1	…	Word_N

The lookup table is created because HBase does not support the secondary index. HBase uses RowKey, column family, and column qualifier as an index of searching by sorting the value of it in ascending order. An index is the mapping of the search keywords to the searched values so that in this case, RowKey can be used as the search keyword, and ColumnQualifier is used as a value. Each index type has a unique prefix defined in the application. Thus adding index types can be done dynamically, and searching by this index is limited in the prefix. This research created four lookup tables to support each searching category by an index of a word or keywords, title, author, and year.

Table 3. Original lookup table in HBase

RowKey				DocumentFrequent				
Word	Title	Author	Year	DocumentId_1	Sum	…	DocumentId_N	Sum

(a)

RowKey	DocumentFrequent				
Word	DocumentId_1	Sum	…	DocumentId_N	Sum
Title	DocumentId_1	Sum	…	DocumentId_N	Sum
Year	DocumentId_1	Sum	…	DocumentId_N	Sum
Author	DocumentId_1	Sum	…	DocumentId_N	Sum

(b)

RowKey	DocumentFrequent				
Word_1 … Word_N	DocumentId_1	Sum	…	DocumentId_N	Sum
Title_1 … Title_N	DocumentId_1	Sum	…	DocumentId_N	Sum
Year_1 … Year_N	DocumentId_1	Sum	…	DocumentId_N	Sum
Author_1 … Author_N	DocumentId_1	Sum	…	DocumentId_N	Sum

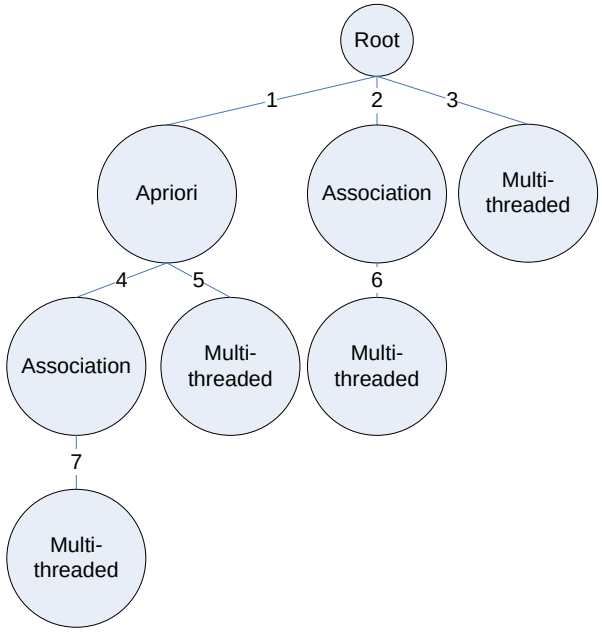
(c)

There are three possibilities of the lookup table design. First is made into an original lookup table (Table 3. a), and the other is combined of all keywords combination, namely combined lookup table (Table 3. b).

Table 3 shows the data model design where the value of RowKey consisted of Word, Name, Year, and Author. The difference between (a), (b), and (c) is the value of the RowKey, where there is one table physically in the (a) model or the number of ColumnName in one ColumnFamily is more than one (the experiment result of this model is as seen in Table 2). Otherwise, there are four tables physically in (b) and (c) model with the value of

each row is consisted one category which is Word or Author or Year or Author or one ColumnFamily has one ColumnName. Besides that, the value of RowKey of the (b) model is consisted of one word, while (c) the value of RowKey of the (c) model consists of an index 1 to N combination of each category. For example, if the total of Word in the Orginal lookup table is 10, the total of words combination is (2N-1) = 1023 records, so there will be the addition of a keyword which is the combination of Word in all of the Keywords.

The (a) model was not implemented in this research, because of the similarity mechanism in structuring data with the basic information table in HBase (Table 3). There were possibilities for one Word located in more than one DocumentID and the value of RowKey is too specific which is consisted of Word, Name, Year, and Author, so the (a) model is contradictive to the Inverted Index idea itself.



(a)

RowKey
Apriori
Association
Multi-threaded
AprioriAssociation
AprioriMulti-threaded
AssociationMulti-threaded
AprioriAssociationMulti-threaded

(b)

Figure 4 The RowKey combination.

Figure 4 shows how the combination is generated for the Word category by Trie concept for 1 document which has three keywords are Apriori, Multi-threaded, Association. The total value of RowKey for it is seven combinations, and there will be seven rows in a table of Word, where each row has the number of DocumentId in DocumentFrequent based on the occurrence of each word combination in the final project document. it aims to speed up the search for a list of documents with only one reading process for keywords that are searched for more than one word. Where in (b) model, searching for more than one keyword will require scanning the table according to the number of words entered.

To make the RowKey list as seen in Figure 4 (b), we have to arrange the data which is entered via a user interface and ordered by a prefix of words from A to Z or lexicographically sorted. There are many techniques to get a document of inverted index cases [4], in this research the trie [13], [14] data structure is used to get a combination of a word that more than one keyword entered by a user. The total table of HBase for the document management system is five tables, namely BasicInformation, WordLookup, TitleLookup, AuthorLookup, YearLookup table. The DocumentId field in all of the tables refers to DocumentId field in BasicInformation table. To keep the similarity of the data model used between HBase and MongoDB, the (c) model, as seen in Figure 3 is selected to match the embedded data model used in MongoDB.

3.2. Data Structure

There are three objects to perform the query in HBase and MongoDB, namely BasicInformation (Figure 8), Document (Figure 5 and 6), and DocumentFrequent (Figure 7). The data structure of each object is as follows:

```
private int _id;
private List<DocumentFrequent> documentFrequent
        = new ArrayList<DocumentFrequent>();
public Document(int node,
        List<DocumentFrequent> documentFrequent) {
    super();
    this._id = node;
    this.documentFrequent = documentFrequent;
}
```

Figure 5 The data structure of Document collection for primary index at _id attribute.

To know indexing performance other than default _id indexing (primary index) in MongoDB, the data variable which was one of the attributes in Document class (Figure 6) was assigned as a secondary index, as seen in Figure 6.

```
private int _id;
private String data;
private List<DocumentFrequent> documentFrequent
        = new ArrayList<DocumentFrequent>();
public Document(int node, String data,
        List<DocumentFrequent> documentFrequent) {
    super();
    this._id = node;
    this.data = data;
    this.documentFrequent = documentFrequent;
}
```

Figure 6 The data structure of Document collection for secondary index at data attribute.

In inverted index application, the documents are displayed in ascending order based on the number of times the keywords appear in the document, as seen in Figure 7.

```
private int document;
private int frequent;
public DocumentFrequent(int document, int frequent) {
    super();
    this.document = document;
    this.frequent = frequent;
}
```

Figure 7 The data structure of DocumentFrequent.

The data attribute in Document class is Word or Title or Author or Year. There are three objects to perform the query in HBase and MongoDB, namely BasicInformation (Figure 10).

```
private int _id;
private String title;
private Date year;
private String author;
private List<String> keywords = new ArrayList<>();
public BasicInformation(int node, String title,
        Date year, String author, List<String> keywords) {
    super();
    this._id = node;
    this.title = title;
    this.year = year;
    this.author = author;
    this.keywords = keywords;
}
```

Figure 8 The data structure of BasicInformation.

3.3. Experimental Results

In this research, the number of keywords was tested was 5 combinations of words using 5K, 50K, and 100K records due to knowing which one is faster between column-oriented (HBase) and document-oriented (MongoDB). Testing was carried out ten times in a single server environment PC with Intel (R) Core (TM) specifications i5-8250U CPU @ 1.60GHz 1.80 GHz uses 12.0 GB RAM, MS Windows 10. The datasets are not generated comprehensively so that they are reached as sufficient data sources for scientific studies in knowing which data model design produces the best time performance.

3.3.1. HBase

Since HBase does not have secondary indexes, so the value of the experimental result will be compared to the best performance with utilizing the block size and block cache of HBase. In addition, Table 2 shows how was run time when the comparison of block size, where the default MongoDB block size is 1MB [15], while HBase is 64KB [5] and the number of ColumnName in one ColumnFamily is more than one. Special dataset (different with the dataset of Scenario_3 and Scenario_4) was generated by system to support the experiment for one column of HBase, where the block size has been configured to be similar with MongoDB that is 1MB as well or 1048576 Bytes. Figure 9 shows the HBase configuration with a block size is 65536 Bytes or 64 Kilobytes and a block cache is 'TRUE', while Figure 10 shows the HBase configuration with a block size is 1048576 Bytes and a block cache is 'TRUE'.

```
COLUMN FAMILIES DESCRIPTION
{NAME => 'Document', BLOOMFILTER => 'ROW', V
ERSIONS => '1', IN_MEMORY => 'false', KEEP_D
ELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING
 => 'NONE', TTL => 'FOREVER', COMPRESSION =>
 'NONE', MIN_VERSIONS => '0', BLOCKCACHE =>
'true', BLOCKSIZE => '65536', REPLICATION_SC
OPE => '0'}
```

Figure 9 HBase configuration for block size is 16KB and block cache is 'TRUE'.

```
COLUMN FAMILIES DESCRIPTION
{NAME => 'Document', BLOOMFILTER => 'ROW', V
ERSIONS => '1', IN_MEMORY => 'false', KEEP_D
ELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING
 => 'NONE', TTL => 'FOREVER', COMPRESSION =>
 'NONE', MIN_VERSIONS => '0', BLOCKCACHE =>
'true', BLOCKSIZE => '1048576', REPLICATION_
SCOPE => '0'}
```

Figure 10 HBase configuration for block size is 1MB and block cache is 'TRUE'

Table 4. Scenario_1: Configuring the block size of one ColumnFamily (Document) becomes 1 and 100 ColumnName.

Records	Block size	Column	Average (ms)
100K	64KB	1	8220
		100	57589
	1MB	1	2187
		100	59379

Table 4 shows that the smaller number of ColumnName created in one ColumnFamily, the faster the run time conducted. Otherwise, the fastest run time occured when the block size value is bigger (1MB) which is 2187 ms than 64KB which is 8220 ms for 100K records.

In order to know how was the effect of block cache and block size configuration of HBase, the experiment was conducted by setting up the value of block cache that becomes 'FALSE', as seen in Figure 11.

```
COLUMN FAMILIES DESCRIPTION
{NAME => 'Document', BLOOMFILTER =>
'ROW', VERSIONS => '1', IN_MEMORY =>
 'false', KEEP_DELETED_CELLS => 'FAL
SE', DATA_BLOCK_ENCODING => 'NONE',
TTL => 'FOREVER', COMPRESSION => 'NO
NE', MIN_VERSIONS => '0', BLOCKCACHE
 => 'false', BLOCKSIZE => '1048576',
 REPLICATION_SCOPE => '0'}
```

Figure 11 HBase configuration for block size is 16KB and block cache is 'FALSE'.

Table 5 shows the result of this experiment, where the run time is taking more time than the previous experiment by setting up the block cache becomes 'TRUE' to improve the performance of reads [16].

Table 5. Scenario_2: Configuring the block size is 1MB and block cache is 'FALSE' for ColumnFamily (Document).

Records	Block size	Column	Average (ms)
100K	1MB	1	4302
		100	149158

The time difference for 100K records and 100 ColumnName is almost three times, where the 'TRUE' value of block cache configuration is more faster which is 59379 ms than 'FALSE' which is 149158 ms.

3.3.2. MongoDB

There are three data model that can be implemented in MongoDB, where the best data model is embedded [12]. Furthermore, there are two types of indexing that examined in this research which are primary and secondary indexing. The primary indexing is in _id field of MongoDB, while the secondary indexing is data field. The comparison results for each indexing types is seen in Table 6, where for secondary indexes is faster than primary indexes which is 1320 ms.

Table 6. Scenario_3: Implementing primary and secondary indexes.

Records	Category	Average (ms)
100K	Primary indexes	1448
	Secondary indexes	1320

3.3.3. HBase vs MongoDB

Figure 12 shows that the best data model for HBase is must utilizing primary indexing with 1MB for block size with consisted 1 ColumnName in 1 ColumnFamily and 'true' value for block cache. Otherwise, the best data model for MongoDB is using secondary indexing. Figure 18 shows that secondary indexes of MongoDB is faster than HBase for 5K records which is 293 ms, but primary indexes of HBase is faster than MongoDB for 50K records which is 533 ms and for 100K records which is 586 ms as well.

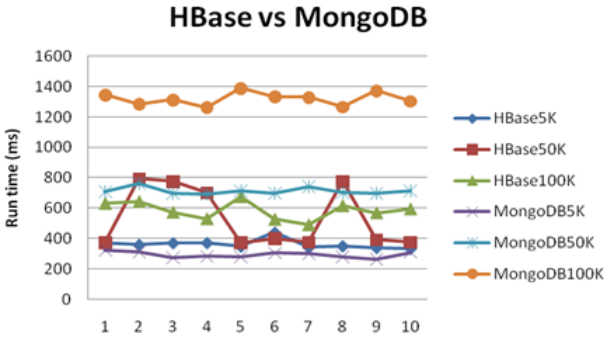


Figure 12 Run time of HBase vs. MongoDB**.**

Table 7 shows detail of the HBase vs MongoDB experimental results.

Table 7. Scenario_4: experimental results of HBase primary indexes with the block size is 1MB and 1 ColumnName in 1 ColumnFamily, and MongodDB secondary indexes implementation for searching Documents by criteria of Keywords of the final assignment documents in milliseconds.

Database	Category	Records	Average (ms)
HBase	Primary indexes	5K	364
		50K	533
		100K	586
MongoDB	Secondary indexes	5K	293
		50K	714
		100K	1320

4. CONCLUSION

This research has obtained performance results in searching for the list of DocumentID and Frequent by using the embedded data model implemented in HBase and MongoDB. Based on the average of experimental result for 5K, 50K, and 100K shows that HBase is faster than MongoDB. The secondary indexes implementation of MongoDB was more optimal than primary indexes

both at 5k and 100K records and the run time for HBase with one ColumnName number was always more faster than 100 ColumnName number contained in one ColumnFamily. Further research is needed to get the performance of getting BasicInformation data by doing a union or other table (HBase) or collection (MongoDB) merging methods based on the DocumentID value that has been obtained. Moreover, the rowkey combination data model (Figure 4) will be interested to explore in another experimental indexing research in HBase and MongoDB.

REFERENCES

[1] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "NoSQL databases: Critical analysis and comparison," 2017 Int. Conf. Comput. Commun. Technol. Smart Nation, IC3TSN 2017, vol. 2017-Octob, no. February, pp. 293–299, 2018, doi: 10.1109/IC3TSN.2017.8284494.

[2] K. Fraczek and M. Plechawska-Wojcik, "Comparative analysis of relational and non-relational databases in the context of performance in web applications," Commun. Comput. Inf. Sci., vol. 716, no. April, pp. 153–164, 2017, doi: 10.1007/978-3-319-58274-0_13.

[3] A. Prasad, "HBase vs Mongo DB," no. December 2018, 2019, doi: 10.13140/RG.2.2.15766.80968.

[4] A. K. Mahapatra and S. Biswas, "Inverted indexes-Types and techniques," Int. J. Comput. Sci. Issues, vol. 8, no. 4, pp. 384–392, 2011.

[5] "Configuring the Blocksize for HBase | 6.3.x | Cloudera Documentation." https://docs.cloudera.com/documentation/enterprise/latest/topics/admin_configure_blocksize.html (accessed Aug. 20, 2021).

[6] X. Gao, L. Hall, S. W. Ave, J. Qiu, L. Hall, and S. W. Ave, "Scalable Inverted Indexing on NoSQL Table Storage," 2010.

[7] C. O. Truică, F. Rădulescu, and A. Boicea, "Building an inverted index at the DBMS layer for fast full text search," Control Eng. Appl. Informatics, vol. 19, no. 1, pp. 94–101, 2017.

[8] "Secondary Indexing | Apache Phoenix." http://phoenix.apache.org/secondary_indexing.html (accessed Sep. 02, 2021).

[9] B. Nair, "HBase Caching & Performance Mo < va < on," no. June, 2014.

[10] "Indexes — MongoDB Manual." https://docs.mongodb.com/manual/indexes/ (accessed Sep. 01, 2021).

[11] "Data Model Design — MongoDB Manual." https://docs.mongodb.com/manual/core/data-model-design/ (accessed Aug. 31, 2021).

[12] A. Hodijah and U. T. Setijohatmo, "Experimental Evaluation of Relationships Model Between Documents in MongoDB," vol. 198, no. Issat, pp. 525–531, 2020, doi: 10.2991/aer.k.201221.087.

[13] M. Terrovitis, S. Passas, P. Vassiliadis, and T. Sellis, "A combination of trie-trees and inverted files for the indexing of set-valued attributes," Int. Conf. Inf. Knowl. Manag. Proc., no. January, pp. 728–737, 2006, doi: 10.1145/1183614.1183718.

[14] J. Fischer, F. Kurpicz, and P. Sanders, "Engineering a distributed full-text index," Proc. Work. Algorithm Eng. Exp., vol. 0, pp. 120–134, 2017, doi: 10.1137/1.9781611974768.10.

[15] "Configure Block Size in a Blockstore — MongoDB Ops Manager upcoming." https://docs.opsmanager.mongodb.com/current/tutorial/configure-block-size/ (accessed Aug. 20, 2021).

[16] S. Saquib, "BigTable vs. HBase," no. April 2015, 2018.