Computer Systems Science & Engineering
DOI:10.32604/csse.2022.020452
Article

Tech Science Press

A Novel Big Data Storage Reduction Model for Drill Down Search

N. Ragavan and C. Yesubai Rubavathi*

Department of Computer Science and Engineering, Francis Xavier Engineering College, Anna University, Tamil Nadu, India
*Corresponding Author: C. Yesubai Rubavathi. Email: yesubairubavathic@francisxavier.ac.in
Received: 25 May 2021; Accepted: 03 July 2021

Abstract: Multi-level searching is called Drill down search. Right now, no drill down search feature is available in the existing search engines like Google, Yahoo, Bing and Baidu. Drill down search is very much useful for the end user to find the exact search results among the huge paginated search results. Higher level of drill down search with category based search feature leads to get the most accurate search results but it increases the number and size of the file system. The purpose of this manuscript is to implement a big data storage reduction binary file system model for category based drill down search engine that offers fast multi-level filtering capability. The basic methodology of the proposed model stores the search engine data in the binary file system model. To verify the effectiveness of the proposed file system model, 5 million unique keyword data are stored into a binary file, thereby analysing the proposed file system with efficiency. Some experimental results are also provided based on real data that show our storage model speed and superiority. Experiments demonstrated that our file system expansion ratio is constant and it reduces the disk storage space up to 30% with conventional database/file system and it also increases the search performance for any levels of search. To discuss deeply, the paper starts with the short introduction of drill down search followed by the discussion of important technologies used to implement big data storage reduction system in detail.

Keywords: Big data; drill down search; storage reduction model; binary file system

1 Introduction

The software utility used to search information on World Wide Web is called search engine. Generally, the search results are called search engine result pages. Gani et al. [1] states that the future search engine big data volume will be in zeta scale. Nowadays, big data has prompted more research that has improved the disk-based systems to support ultra-low latency service. Existing search engines lack the category-based search feature and drill down feature. Hence the end users need to navigate the page to find the relevant web page.

If a Search Engine has more than one level of search capability, then it is called Drill down Search Engine. Drill down search engine often enhances the accuracy of getting search results. Category based search engine's file system size cannot be of same standard and it can be varied from small, medium, big and very big size based on the search term and its category. The majority of existing systems such as

Google GFS, HDFS or Blobseer use a chunk file size of 64 MB [2]. Normally top search keyword to page relationship is maintained in the search engine file system model. For drill down search, the search time is directly proportional to the N levels of search. Also category based search feature will increase the number and size of the file system model, because a keyword may belong to n-categories.

The vital process in drill down search is the choosing of correct format of file system which offers storage efficiency and fast I/O operations. Classical RDBMS is not suitable for search engine operations. Various frameworks like Spark have reduced big data storage. The Apache Spark framework is much faster because of its in-memory storage and distributed computation. Parquet and Avro file systems are used in Spark frameworks which are very fast and efficient for big data analytics [3,4]. The data is represented as a binary format in Parquet and Avro. This makes it easy to read and write by any program. It also supports batch processing.

This paper proposes an efficient file system model like Parquet and Avro that aims at minimizing disc storage, inter-node communication cost and that can facilitate building scalable drill down support search engine. This novel big data drill down file system would greatly enhance the performance of search engine for any level of searching and it would increase the performance computing of a search engine. The implementation details and experimental reports are also provided in detail. The prototype of the drill down search user interface with category support is shown in Fig. 1. The rest of the paper is organized as follows. Section 2 offers a literature review. Section 3 presents problem statements. Section 4 describes the research design with subsections on keyword header file, keyword data file and N-level drill down search. Section 5 describes the experimental results. Section 6 contains the conclusion.

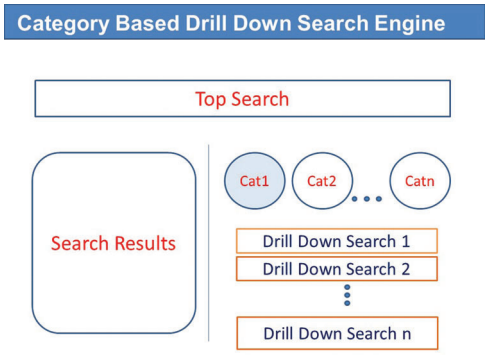

Figure 1: Drill down search engine layout

2 Related Works

A lot of work has been done in the field of Search Engine Big Data Storage and Search Optimization. However, till now no work has been done relating to multi-level search feature. Previously we have conducted so many benchmarks with different file formats as well as variety of databases staring from relational to No-SQL with different data structures in C++ language for drill down search. We have identified that when the size of the search results increases, more redundant data are required to be stored to achieve the drill down search feature with minimum search time. If we intend to minimise the data storage, the drill down search time increases drastically when the number of search level increases.

Disc based analytics offers I/O latency issue. Antaris et al. [5], point out that In-memory data analytics are much faster than disc based analytics and has almost no latency issue. He has also identified that many big data processing frameworks have been evolved based on In-Memory Map Reduce paradigm such as Apache Spark, Apache Storm and Apache Flink.

Puangsaijai et al. [6] compare the performance between No-SQL databases with Redis (Memory DB). The author states that depending upon the situation one could determine which one is better and the performance between each of them can vary depending upon the project nature. They have conducted experiment on insert and delete operation in the same dataset. In [7] authors Mahajan et al. have mentioned that No-SQL database such as Mongo DB does not know which one should be tuned up for quick look up after the documents are formed as a collection. Hence querying gets slower. Fetching records from the huge Mongo DB is lagging because of the absence of schema. Thus, mongo DB cannot be served as a good platform for search term query batch process.

Jatakia et al. [8] have implemented an exhaustive search algorithm which retrieves the stored data from the main memory. The author loaded all the data at a maximum of 320 MB into the memory at once. The delay for computing Hamming distances for a single query is 2.6 s, and is slower than higher language data structures. This also leads to consistency issues between the memory data records. But sometimes it is unable to keep too big data in main memory in Redis. In such cases, selective data that need to be processed are kept in main memory in n times and finally all the resultant data are aggregated.

Keeping much iteration in main memory also takes additional time to search the relevant data. In such circumstances, Parallel computations play a vital role in multi-level computation. The author Strohbach et al. [9] state that distributed file system such as HDFS has been designed for large data files and very well suited for bulk processing. Yu et al. [10] have revealed that parallel data computation using Hadoop could be easy to enable developers to leverage hundreds of nodes to process data. But the authors Liu et al. [11] have stated that Hadoop has issues with small level data and it supports batch process only.

Fangzhou et al. [12] have implemented a POSIX-compliant distributed file system for large data that provides a high customizable Map Reduce view for indexing. The authors demonstrated that for generating high efficiency file system for multi-level searching, CouchDB can be used. It functions like multi-level directory. It seems that performance for CouchFS in searching need external script code that run like batch process to index such large data. Chen et al. [13] stated that in order to use a distributed system to store massive data, the factors such as Consistency, Availability and Partition Tolerance should be taken into consideration. Hence we have decided to implement a custom file system that support multilevel search as well as low disk space.

The author Bhat [14] has investigated the various factors that had led to big data storage capacity gap and pointed out the problems due to data compression. Compression and decompression could not be a better solution for efficient data storage. The author also demonstrated that ultra-storage density and ultra-high throughput of optical storage could meet the demands of big data storage.

The authors Najafabadi et al. [15] stated that Binary codes require little storage space, and they allow quicker searches by computing the Hamming distance between two binary codes. They have also stated that using binary system for information retrieval is more accurate and faster than other methods.

Based upon the above literature survey, we have decided finally to use a distributed custom BINARY FILE SYSTEM for drill down search feature in search engine.

3 Problem Statements

The bulk data of search engine is the main threat for implementing drill down feature. Batch pre-processing technique improves the speed of data retrieval generally. But batch pre-processing could not be implemented because of search engine big data. That is because so many combination of search keywords can be made into account in batch pre-processing which need high end nodes to process and it takes a very long time for indexing. Thus a rigours mechanism is required to generate a file system that uses less data in main memory for faster multi-level searching with the minimised disk space occupancy.

4 Research Design

Normally in a web search, 1 to 1 relationship is found that is keyword to webpages. Whenever a user searches any keyword, its corresponding web pages are listed as search results. So in our research, we have saved the keyword to pages relation in the form of binary file. We know that in binary file, the data are arranged as a sequence of bytes. The advantage of binary file is that it can be directly loaded into the main memory with the help of basic array structure in any language. Also loading of binary file into main memory will take very less time when compared to any other file format. Moving the pointer to the required segmentation is very fast in binary data. Due to this reason, we have chosen binary file format. In our proposed system, we have written 2 binary files in crawling process for drill down search. Fig. 2 shows the binary files create work flow in crawler batch. In the crawler batch process, the crawler program downloads the web page content and parses the keywords. Keywords found in Page Title have high rank value wheresas Keywords found in the page URL, Meta tag and main content have high, fair and low values. Keywords that have highest rank value are considered as a Category.

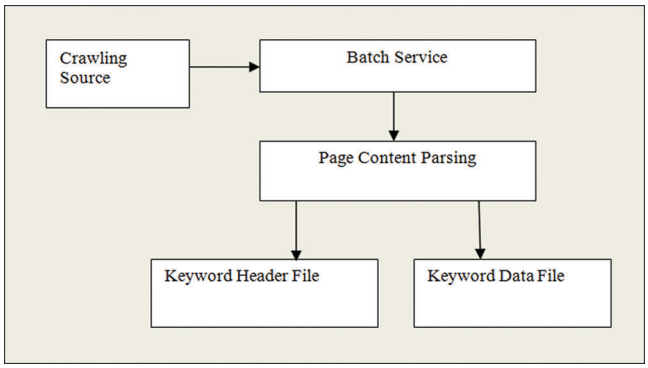

Figure 2: Keyword binary files create flow in crawler batch process

The Category selection formula is

Category = $\sum_{i=1}^{n} R(K)$ where R (K)—Rank of the keyword, n—No of occurrences of the keyword

Two types of binary files are created per category in the crawler batch. They are

1. Keyword Header File
2. Keyword Data File

4.1 Keyword Header File

Header file contains the Meta data of pages. This file size is small and it contains the offset details of data file. Fig. 3 shows the Header File Format. The first 32 bit data represents the total number of unique keywords found in a category search. If n would be the number of unique keywords found, then after first 32bit, n*32 bits represents the n keyword ids respectively. After n*32 bits, next n*32 bits represents the keyword's page offset details. These keyword page offset refers the page ids in the Data file. Since the Header file size is small, it can be loaded into main memory quickly. Also header file can be scanned periodically as a background process very easily.

KEYWORD COUNT	KEYWORD ID DATA	KEYWORD OFFSET DATA
32 bit	32 bit * No. Of Keywords	32 bit * No. Of Keywords

Figure 3: Keyword header file format

Header File Example

Let us assume that 5000000 keywords are available for a category. Let us assume the first 3 keywords as follows.

First keyword Name: Trump. First keyword ID: 111.

Second keyword Name: Family. Second keyword ID: 222.

Third keyword Name: Ivanka. Third keyword ID: 333.

The first three keywords and its binary representation are shown below.

Like this way, remaining keywords are arranged in the header file.

Then we need to represent how many pages belong to each keyword.

First Keyword Page count: 1430000.

First Keyword Offset: 1430000.

Second Keyword Page count: 58000

Second Keyword Offset: 1430000 + 58000 => 1488000.

Third Keyword Page count: 32127

Third Keyword Offset: 1488000 + 32127 => 1520127.

Like this way, remaining keywords and its offset details are maintained in the Header file.

Header File Size Calculation

Let the total keyword be n. Then the size of the header file is

Keyword count size = 1 * 32 bits => 32 bits

Keywords size = n * 32 bits

Keywords offset size = n * 32 bits

Total size = (32 + 2(n * 32)) bits.

Then as per our example, the size of the header file is (32 + 2(5000000*32)) => 320000032 bits

=> 40 MB.

Thus 40 MB is required to load this header file into main memory.

Also the loading time of 40 MB into main memory is a few milliseconds.

Fig. 4 shows the Header file layout and its binary representation. From this figure, it is very clear that data written in binary format are arranged as a sequence of bytes.

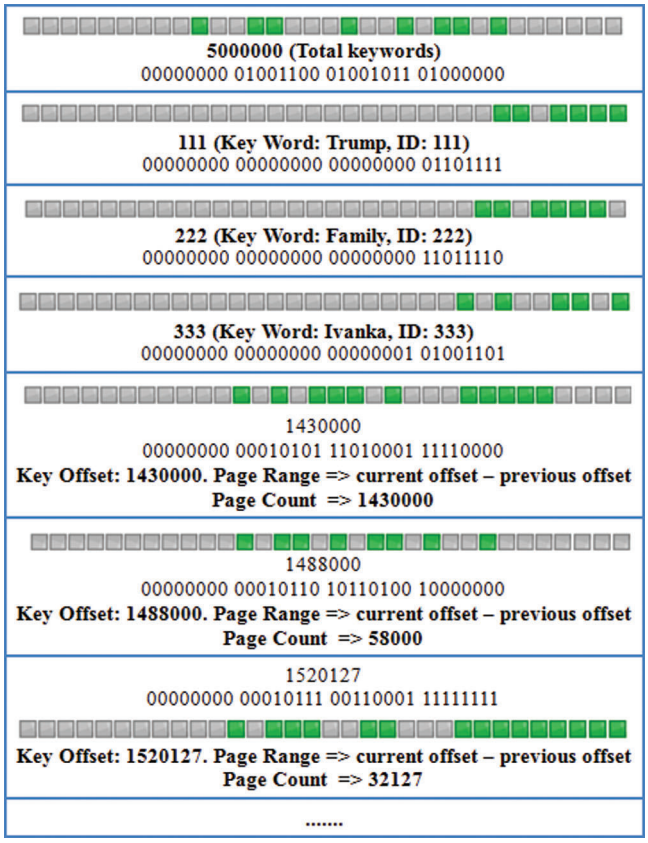

Figure 4: Keyword binary file layout

The algorithm for writing and reading keyword header file is shown below. The size of the keyword is selected as 32 bit. Since we are using category based searching, keywords are distributed among categories. So unique keywords count per category gets limited. During crawling, if the keyword id reaches the maximum 32 bit level, instead of using 64 bit for keyword, newer category is generated in the crawler. As keyword header files contain the offset position of keyword data file, both the files are generated simultaneously.

Algorithm 1: Algorithm for Writing Header File

Input: Key ID with Pages Data Set T,

Output: Binary Header File

Start

Read T

Open Header File *F

Find the Key ID Dataset Length as n

Write n as Keywords count in *F

For each KeyID record in T as r

Write keyword record r in *F

End

(continued)

Algorithm 1: (continued)

```
        For each KeyID record in T as r
            Initialize pre_offset as 0
            Fetch list of Pages as P from r
                Keyword_offset of = (P count) + ( pre_offset)
                Write offset record of in *F
                Assign P count to pre_offset
        End
        Close Header File *F
Stop
```

Algorithm 2: Algorithm for Loading Header File into Main Memory

Input: Binary Header File

Output: Header File loaded into RAM via keyword Array, Offset Array

Start

Open Header File *F

Read 32 bits and assign as an Integer data type n (keyword count)

Read 32*n bits and assign to a dynamic array (keyword array)

Read 32*n bits and assign to a dynamic array (offset array)

Close Header File *F

Stop

4.2 Keyword Data File

This binary file contains the Page Ids of Header file keywords. The data file format is shown in Fig. 5. Data file size may reach up to Gigabyte size due to the fact that data are redundant in nature. The size of the file greatly depends on the Header File keywords count. As the size of the file is big, it is not loaded into the memory and instead, the data offsets are fetched from main memory. Since in-memory data access is very fast when compared to disk based access, keyword offset in header file is used here as an efficient index. By using this keyword offset, required page results are fetched from the data. In order to further improve the data file disc access speed, SSD drives are preferred to store data file instead of HDD. The authors Zhang et al. [16] have reported that quick accessing the big data is a major challenge and SSDs (Solid State Drives) would rectify this issue. It can access data faster than HDD.

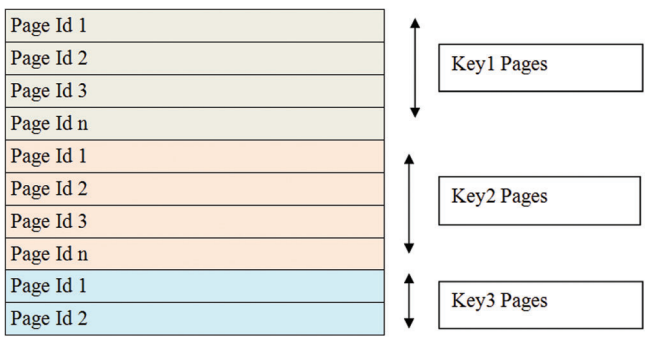

Figure 5: Keyword data file layout

Algorithm 3: Algorithm for Writing Data File

Input: Key ID with Pages Data Set T,

Output: Binary Data File

Start

Read T

Open Data File *F

For each KeyID record in T r

Identify list of page ids as *p

write *p in *F

End

Close Data File *F

Stop

4.3 N-Level Drill Down Search

4.3.1 Main Search

As the keyword header file is loaded in the main memory, when an end user searches any keyword in the main search, the search keyword's ID need to be fetched at first from a hash structure (which is created and maintained in crawl batch process). After getting the Keyword ID, its corresponding offset details are fetched from the main memory. By using the offset, page Ids are fetched from keyword data file. Normally fetching keyword offset from main memory will take around 2 to 5 milliseconds whereas fetching from data file will take around 20 to 50 mill-second even though the size of the data file is in GB. Fetched main search page ids are converted into the pages' short text in the user interface side. During this span of time, main search page ids are temporarily cached in the server in the form of key-value dictionary object in parallel.

4.3.2 Level 1 to n Search

When the end user searches any keyword in Level1, the same main search process is repeated in which the resultant page ids are intersect with those one available in the temporary cache in the server. The resultant pages are then sent to the end user. In the server, cache page ids are removed from memory and the Level1 resultant page ids are now stored in the server cache. This same process gets repeated when the user searches in the upcoming levels. Fig. 6 depicts the entire process.

In case, enough server memory is not available for cache process, every keyword pages should be kept in a dynamic array structure during drill down search. Then intersection of these dynamic array structures should be done such that the order of intersection is based on keywords having fewer pages. This will reduce the intersection time. Fetching the keyword pages from binary file time is in the order of milliseconds. The time consuming process in drill down search is the intersection process only. Selecting the correct data structure will speed up the drill down search.

In our proposed drill down process, we have selected unordered map data structure and vector data structure in C++. The keyword page count is written in the binary header file. As already header file is available in the server main memory, when a drill down search request is sent to the server, within a millisecond, each level keyword's page count is fetched. Now, the drill down level keyword having less page count is pushed into unordered map structure. Other level keyword pages are pushed into vector data structure. These other levels vector data structures are iterated and compared with the unordered map structure and common pages are fetched. This type of intersection process is very quick.

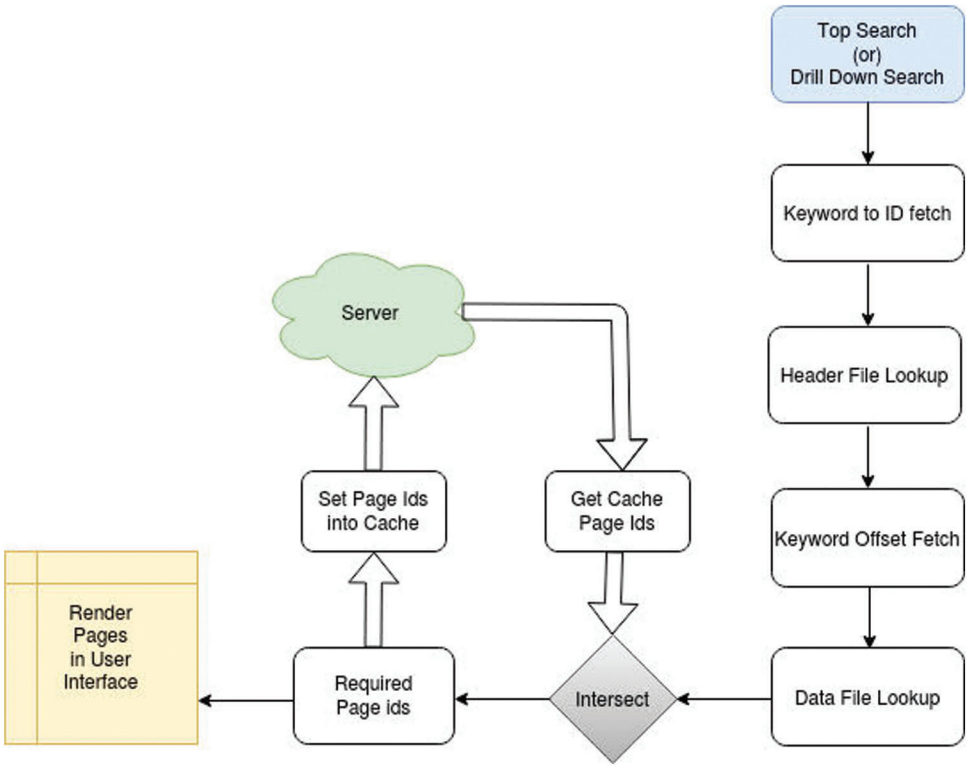

Figure 6: Drill down search flow

Higher level of drill down searching leads to more accurate search results. Our proposed drill down file system is based on categories. Normally for each category, one header and one data binary file are created during crawling process. Existing search engines maintain their data in a junk of 64 MB file. Their keyword data may belong to any number of files in the existing system. In our proposed system, we have reduced the size of the binary files using categories. So loading the header binary file into server main memory and fetching the pages from data binary file become very fast. This would enhance our drill down process.

The search results accuracy of our drill down search is superior to existing search engines' top level search (by combining all drill down level keywords into a whole word and search). Thus future generation of search engines definitely would be category based as well as drill down based.

5 Experimental Analysis and Results

The specification of the environmental setup used is as shown in Tab. 1.

Table 1: Experimental system configuration

Environment	Configuration
Processor	Intel i5 10^{th} generation
RAM	16 GB
Operating system	Ubuntu 16.04
Compiler	C++ 11
SSD	Samsung SSD 1 TB

We have conducted so many experiments using C++ to create binary for different sized records. Even for creating 5 million records as binary file, it takes only 19.5 MB. The size of the binary file is exactly proportional to the record count. Even if the record's size increases drastically, the expansion ratio of the binary file is in constant ratio. This binary file feature is greatly used to predict the future disc space requirements. Tab. 2 shows the dataset used to create binary file and Fig. 7 shows the size of the binary file with respect to different sized records. Inserting records into binary file is always performed in the order of milliseconds whether we insert small amount of data or bulk data.

Table 2: Search time data set using binary file

No. of records	Binary file size in Kb
1000	3.9
10000	39
100000	390.6
1000000	3906
5000000	19531.25

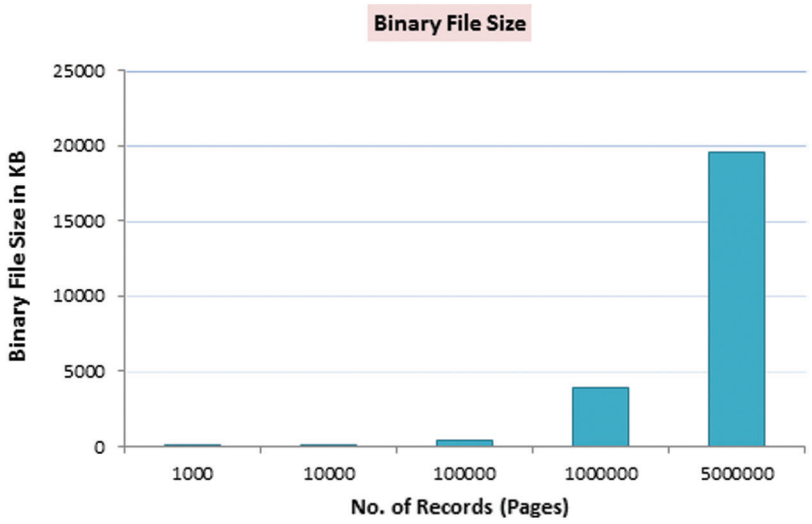

Figure 7: Binary file size *vs.* Number of records

Also fetching records from binary data is always fast if we know the start and end segments of the binary file. The start and end segment positions (here called as keyword offset) are used to fetch the records very fast and it is used to index the data file. Tab. 3 shows the data set that are inserted in the binary file and its corresponding time taken. It is depicted in Fig. 8. Even for inserting 5 million records into binary file, it takes only 73 milliseconds which is too small when compared to the time taken to insert integer data type in any other conventional file systems.

In our experiment, we have used a crawled news category data source (If the crawler is crawling news based articles such as CNN, then the category would be News). This data source consists of 5 million unique keywords. Normally news based articles and Wikipedia articles have huge number of keywords. So news level category dataset is very much useful to predict the load test and search test performance of search engine. The drill down dataset used is shown in Tab. 4.

Table 3: Binary file size data set

No. of records	Insert time in ms
1000	3
10000	3
100000	8
1000000	15
5000000	73

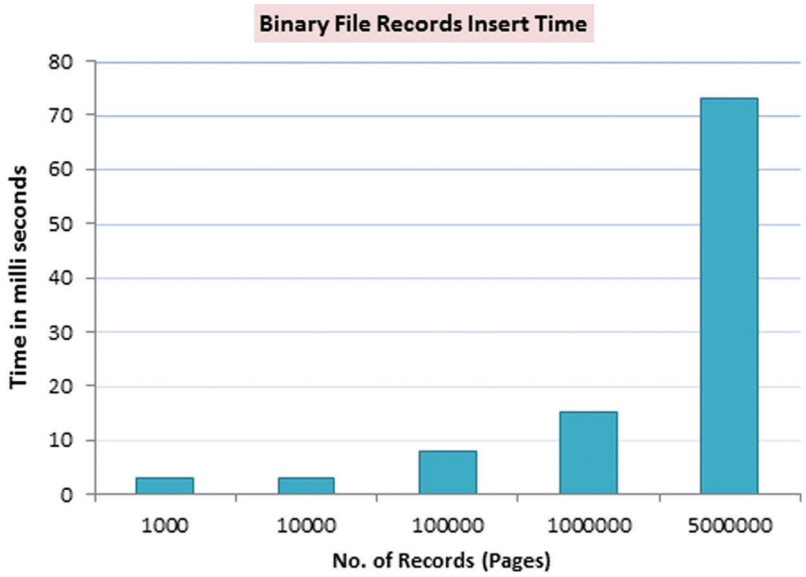

Figure 8: Binary file records insert time

Table 4: Drill down search data set

Keyword	Search level	Key pages count	Inner search pages count	Search time in millisecond
Trump	Main search	1430000	1430000	55
Family	Level 1	58000	47324	60
Ivanka	Level 2	32127	28956	65
Business	Level 3	7546	2156	82

The top search keyword is “Trump”. This is main search. The number of web pages belongs to this keyword is 1430000. The search time in server is 55 milliseconds. At the time of rendering results in the user interface, these 1430000 pages are kept in server cache. In drill down Level 1 search, when the keyword “Family” is searched, it needs to search inside those 1430000 pages. The number of web pages belongs to the keyword “Family” is 58000. After getting the common pages between 1430000 pages and 58000, we have 47324 resultant pages.

These pages are now rendered in the user interface. The Level 1 search time including the intersect process takes only 60 milliseconds. Now the 47324 pages are kept in server cache and the old cache is

removed if no session is used. Similarly for Level 3 and Level 4 search, the search time is around 65 and 82 milliseconds respectively. If we search more and more levels, the search time increases slightly. The main search proposed user interface is shown in Fig. 9. Drill down level 1 search interface is shown in Fig. 10. Drill down level 2 search interface is shown in Fig. 11. Drill down level 3 search interface is shown in Fig. 12. The various search levels and their corresponding search timings are shown in Fig. 13.

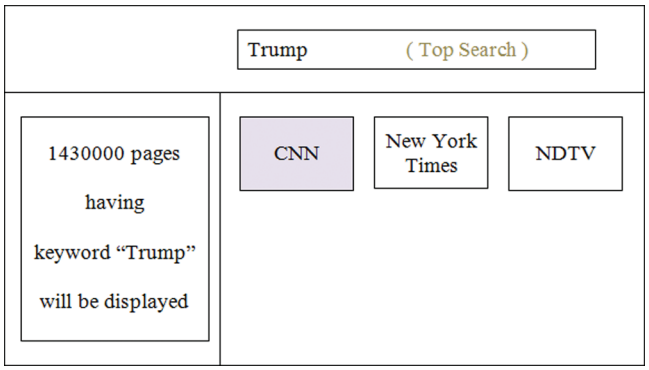

Figure 9: Main search proposed user interface

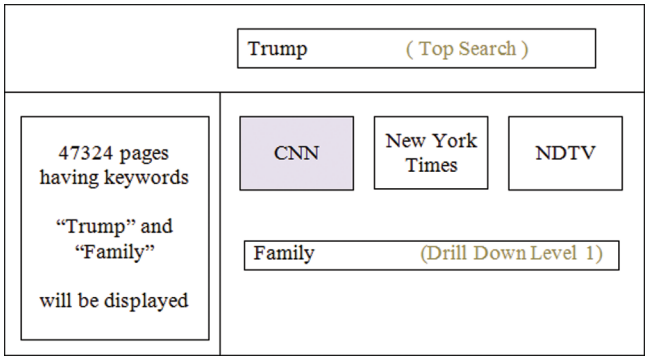

Figure 10: Drill down level 1 search proposed user interface

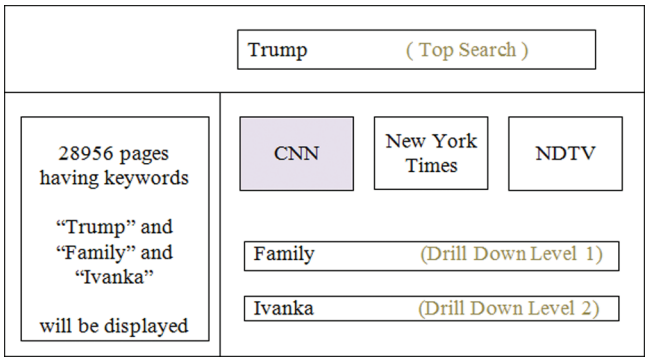

Figure 11: Drill down level 2 search proposed user interface

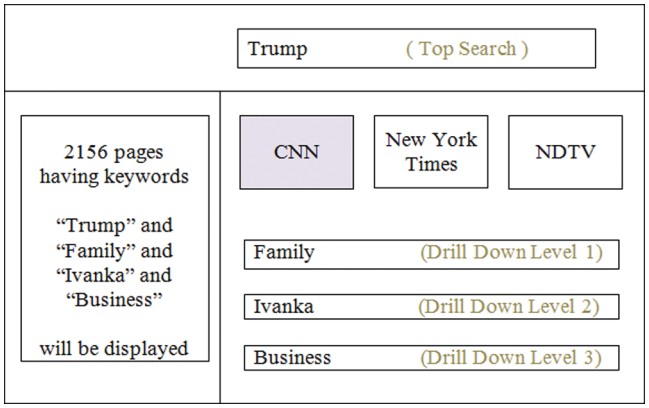

Figure 12: Drill down level 3 search proposed user interface

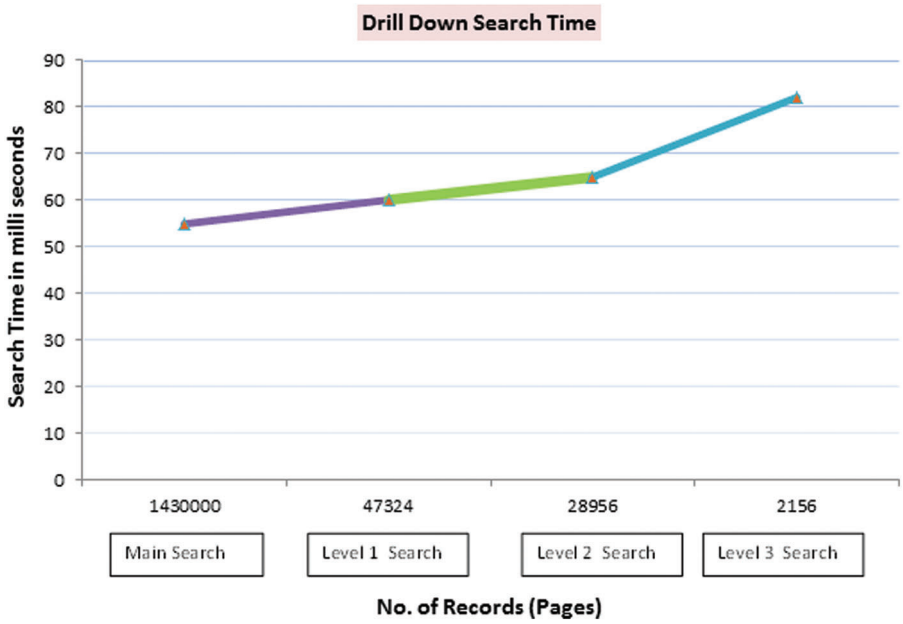

Figure 13: Drill down search time for different levels

6 Future Work

Since binary file format is not readable to human, much care should be taken while writing data into a binary file. Missing a single bit leads to wrong data insertion. Also care must be taken while reading the correct offset position from binary file. Small change in the file pointer position will lead to throw big exception and wrong data. So in our future work, we are studying the possible chance of errors produced when reading and writing data in the binary file. As search engine crawler batch and search routine tools are running continuously, continuous learning about the correctness of the binary entities is important. So in our future work, we are going to create deep learning models using our drill down binary data with the help of Google Tensor flow framework. Deep Learning is more powerful to resolve data analytical and learning problems found in huge data sets and helps to extract the complex data representations from a large raw data [17]. Due to the increased volumes of drill down binary data, applying deep learning will train more binary data in future that will increase the drill down files accuracy.

7 Conclusion

We have implemented the drill down feature to the search engine such that it uses binary file system to store the data which not only reduces the disc space requirement but also increases the search performance for any levels when compared to conventional file system and other data base models. Also by using this file system we can easily predict the future disc requirement in crawler batch process. The important feature of binary file logic is that it can be moved to main memory in a fast manner. Fetching the data from the binary file could be very fast and we proudly say that our Binary File System model would act as an efficient storage reduction model for drill down search.

Funding Statement: The authors have not received any fund for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Gani, A. Siddiqa, S. Shamshirband and F. Hanum, "A survey on indexing techniques for big data: Taxonomy and performance evaluation," *Knowledge and Information Systems*, vol. 46, pp. 241–284, 2016.

[2] A. Elomari, L. Hassouni and A. Maizate, "The main characteristics of five distributed file systems required for big data: A comparative study," *Advances in Science, Technology and Engineering Systems Journal*, vol. 2, no. 4, pp. 78–91, 2017.

[3] V. Belov, A. Tatarintsev and E. Nikulchev, "Choosing a data storage format in the apache hadoop system based on experimental evaluation using apache spark," *Symmetry*, vol. 13, no. 195, pp. 1–22, 2021.

[4] M. J. Awan, M. Shafry, H. Nobanee, A. Yasin, O. I. Khalaf *et al.*, "A big data approach to black Friday sales," *Intelligent Automation & Soft Computing*, vol. 27, no. 3, pp. 785–797, 2021.

[5] S. Antaris and D. Rafailidis, "In-memory stream indexing of massive and fast incoming multimedia content," *IEEE Transactions on Big Data*, vol. 4, no. 1, pp. 40–54, 2018.

[6] W. Puangsaijai and S. Puntheeranurak, "A comparative study of relational database and key-value database for big data applications," *2017 International Electrical Engineering Congress (iEECON)*, Pattaya, Thailand, pp. 1–4, 2017.

[7] D. Mahajan, C. Blakeney and Z. Zong, "Improving the energy efficiency of relational and NoSQL databases via query optimizations," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 120–133, 2019.

[8] V. Jatakia, S. Korlahalli and K. Deulkar, "A survey of different search techniques for big data," *Int. Conf. on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, India, pp. 1–4, 2017.

[9] M. Strohbach, J. Daubert, H. Ravkin and M. Lischka, "Big data storage," *New Horizons for a Data-Driven Economy.* Springer, vol. 1, pp. 119–141, 2016.

[10] S. Yu, M. Liu, W. Dou, X. Liu and S. Zhou, "Networking for big data: A survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 1, pp. 531–549, 2016.

[11] Y. Liu, Y. Zeng and X. Piao, "High-responsive scheduling with map reduce performance prediction on hadoop YARN," *2016 IEEE 22nd Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Daegu, Korea (South), pp. 238–247, 2016.

[12] Y. Fangzhou and H. Roy, "CouchFS: A high performance file system for large data sets," *2014 IEEE Int. Congress on Big Data*, Anchorage, AK, USA, pp. 784–785, 2014.

[13] M. Chen, S. Mao, Y. Zhang and C. M. Leung, "Big data-related technologies, challenges and future propspects," *Springer Briefs in Computer Science*, vol. 1, pp. 35, 2014.

[14] W. A. Bhat, "Is a data-capacity gap inevitable in big data storage?," *Computer*, vol. 51, no. 9, pp. 54–62, 2018.

[15] M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald *et al.*, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no.1, pp. 1–21, 2015.

[16] H. Zhang and G. Chen, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.

[17] A. Oussous, F. Benjelloun, A. Lahcen and S. Belfkih, "Big data technologies: A survey," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, pp. 431–448, 2018.