DISTRIBUTED INDEXING: PERFORMANCE ANALYSIS OF SOLR, TERRIER AND KATTA INFORMATION RETRIEVALS

[1]Ali Y. Aldailamy, [2]Nor Asila Wati Abdul Hamid and [3]Mohammed Abdulkarem*

[1,2] Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Selangor, Malaysia
[3] Faculty of Engineering, Universiti Putra Malaysia, Selangor, Malaysia

Email: a.aldailamy7@gmail.com[1], asila@upm.edu.my[2], memo.ye@gmail.com[3]

DOI: https://doi.org/10.22452/mjcs.sp2018no1.7

ABSTRACT

Information Retrieval (IR) systems are currently facing a continuous challenge due to the increasing size of datasets. Extremely, large data from different aspects is gathered each day resulting in huge increase in the scale of the raw data available across the internet. Indexing, as the main function of IR systems, is becoming a time-consuming problem. Therefore, efficient indexing of a large volume of data is now a critical requirement of modern IR systems. High performance indexing is performed nowadays over the use of MapReduce programming model. MapReduce is a programming paradigm that enables massive processing and large collections distribution across multiple (hundreds or thousands) commodity computers. To shed some light on this issue, this paper presents a detailed performance analysis of distributed indexing for Solr, Katta and Terrier with the context of MapReduce. In particular, this study compares and analyzes the distributed indexing performance of three frameworks using 1GB, 3GB, 6GB, and 9GB subsets of TREC dataset as the processing power increase. The experiments measure the indexing average time, then throughput, speedup, and efficiency of indexing process. The results show that, Terrier performance is the best in the presence of large collections and scalable processing power. While, Solr performance is the best when having limited computing power and small document collections. Finally, the experimental results show that, Katta produced the worst indexing average time among the three frameworks but its speedup scales linearly with processing power and collection size.

Keywords— Indexing, Information Retrieval, Hadoop, Katta, MapReduce, Solr, Terrier.

1.0 INTRODUCTION

The explosion of information through internet causes the web to be the largest repository of documents. Despite of that, only a very small amount of information can meet the user requirements. This causes the potential challenge for Information Retrieval (IR) platforms in the web. To avoid this, indexing is the mechanism that allows for efficient information retrieval from huge collections. The implementation of the indexing mechanism produces a data structure that is called indexing structure which allows to find the required information without searching all the documents. Blanco et al. [7] outlined that indexing structure is always stored in a storage device in order to improve the search process. Almost all IRs build their index structures based on the inverted index [2] [6]. Inverted index is a mapping data structure that relates the terms to the documents containing them. This structure is formatted as posting lists of the term and all the document IDs it belongs to. Each posting is represented inside the list as an integer (Doc-ID) that refers to the document where the term exists. The posting can also contains more information about its term such as terms occurrence number in the document and the term location in document.

Biswas et al. [56] has stated that, there are three types of data structures for the inverted index namely: Sorted Array, Hashing, Search trees. Each of these data structure has its own advantages and disadvantages in terms of manipulation and lookup. The sorted array structure has the advantage of binary search if it is in memory. However, insertion of new terms can cause overhead. The advantage of hashing structure is that it provides fast look-up, whereas its drawback is that collisions may happen. Tree is the best data structure for building index. That is because it maintains balance and allows for insertion and deletion. All of these structures create posting list for each term in the document. Each of these structures has the ability of mapping the key/values pair. However, each of them delivers different ratio of performance in terms of lookup, insert, and delete [55].

It is a time-consuming process to build this posting list for each term in the collection documents. To overcome this problem by reducing the time needed to index large collections, it is highly recommended to distribute the indexing process with the context of MapReduce across many processing nodes [3]. Although, most of the available indexing frameworks can work on distributed environment with the context of MapReduce, they always deliver different ratio of performance. Thus, it is necessary to decide which system provides better performance with regarding to available scalability and data size. At such scale, the ability to index a huge amount of data while achieve high throughput is always of much value. In spite of the fact that there are several IR frameworks with different indexing structures and strategies, Solr, Katta and Terrier IRs are the most famous indexing frameworks among researchers and in business. All these indexing frameworks can be integrated with Hadoop MapReduce to achieve a high throughput of indexing process by distributing the indexing work across many computer machines [4][5][6].

Therefore, Scalability is one of the important features in distributed environment; it means that by adding more nodes to the cluster, throughput can increase per unit of time. The distributed algorithm Execution time depends on many factors such as architecture, number of processing nodes, and dataset size. Thus, the distributed system scalability is a measure of its capability to achieve higher speedup as the number of nodes is increased. However, the efficiency drops with an increasing number of processing nodes, unless the size of the dataset is increased also. From the above discussion, there are relationship between throughput, speedup, and efficiency. These relationships between all of these factors will be discussed in detail in the experiment section.

Until now, there are several indexing frameworks that can work in distributed environment, but it is difficult to understand the tradeoffs between these solutions, and it is also quite challenging to decide the best solution for a specific task. Therefore, this paper focuses and reviews indexing strategies for the two following purposes: providing the researchers and practitioners with guidance that help in choosing the appropriate indexing strategy, as well as providing comparison and analysis of technical characteristics of the prominent indexing strategy.

Solr, Terrier, and Katta can support the distribution of the indexing process by integrating their frameworks with Hadoop distribution system. In addition to their support for many search techniques, they can index a wide range of file types such as documents of MS Office, XML, and PDFs files. Hung [5] illustrated that Solr is built on top of the Lucene library that is the most famous library for building the inverted index [4]. The most advantage of Solr is its ability to index and retrieve data from unstructured, semi-structured, and structured data. Furthermore, Atreya et al. [9] demonstrated that Solr supports the manipulation of the already created index structure. This allows the IR system to avoid the process of rebuilding the index structure when there are changes in the collection. For Terrier, posting lists of the inverted index structure are built in the memory based on the single-pass indexing algorithm. The aim behind the development of Terrier was to introduce an IR system that supports efficient indexing and effective information retrieval [10]. Recently, Terrier has been improved and enhanced to overcome the new obstacles emerged with the search operations of real time systems. The most advantage of Terrier is its ability to create a compressed in-memory index structure that allows for better utilization of the available memory. Whereas, Katta is introduced as an indexing framework that uses Lucene library in order to add the value of distributed indexing. It is designed to distribute the Lucene indexing process among multiple nodes to decrease the time needed for indexing large collections. Thus, Katta is capable of indexing very large volume of data as index shards on many servers.

The advanced indexing strategies that have introduced by Solr, Katta and Terrier are widely deployed and used by researcher and in many organizations for the purpose of accomplishing the index process with the context of MapReduce. Thus, this study introduces a detailed comparison of their indexing strategies. It presents also a complete evaluation and deep analysis of their distributed performance using the freely existing Hadoop MapReduce. The average time consumed to index the same datasets is used in the performance analysis as well as the throughput achieved and the speedup gained as the cluster size increases. The efficiency of each indexing strategy will be calculated as well. The experiments are conducted on existing frameworks of Solr, Terrier, and Katta through the use of 1GB, 3GB, 6GB, and 9GB subsets of standard TREC dataset.

This paper is an extension of work originally presented in Fourth International Conference on Information Retrieval and Knowledge Management [1]. We add one more indexing framework and more analysis is also provided for the three indexing strategies. The remainder of this paper is structured as follows: section II describes the work related to this study. A brief overview of the architecture and an explanation of the indexing strategy of the frameworks used in this research are provided in section III. Section IV describes the experiments setup and provides the results discussion. Finally, the conclusions and future work of this study are presented in section V.

2.0 RELATED WORK

In the domain of World Wide Web (WWW), the searching operations are performed by the public web search engines such as Yahoo, Bing, or Google which are provided for searching and retrieving documents from the WWW. However, it is impossible to utilize such public search engines to search and retrieve non-public documents of organizations. Either these organizations develop their own internal search engines to be used by organization users, or they use the available search engines libraries and frameworks such as Sphinx [34], Xapian [36], Zettair [35], Indri [33], Solr [54], Terrier [12], and Katta [45], ElasticSearch [47]. As an example, Planetary Data System (PDS) of NASA uses Solr to store and search for specific datasets, instruments, missions, targets, and host information. Furthermore, JSTOR contains collection of documents for more than 50 disciplines of more than 1,400 digital academic journals. JSTOR utilizes Solr to provide its navigation and search operations by allowing the users to search and access all the archival [25]. Every search engine library or platform has its own highlights that make it completely different from others. They also have different strategies of building the index in ways that take into consideration efficient management of the documents [11].

The organizations have two options for choosing the search engine in order to support the decision-making, either the commercial search engine or open source search engines. The commercial search engine is not really practicable choice due to the high fees and their predefined functionality which unable to be modified according to the requirements of organization. In contrast, open source search engines can provide same features or sometime more and it is also modifiable to meet the requirement of the organization. Importantly, they are free and actively maintained [26]. There are several characteristics that should be taken into consideration in order to choose the appropriate open source search engine that can satisfy the organization requirements. The following are the most important characteristics, the type of index structure used by the engine (such as inverted index, database index, or other file structures), searching functionality (such as fuzzy search, Boolean Operators, use of stemming), types of files supported for indexing (for example, HTML, PDF, plain text, and Microsoft Excel, PowerPoint and Word documents), ranking method, and the most essential characteristic is how fast the search engine capable to index documents.

Open source search engines can be categorized according to the programming language, types of supported file formats, flexibility of search, platform support (new versions and documentation), and also the distribution of its operations among multiple machines, scalability, and the flexibility of the ranking. Table I provides a comparison of the most important characteristics of the most prominent vertical indexing open source search engines. The information that are in the form of rating, scaling, and ranking are collected from [50] [51] [52] [11], in which they stated that the rating, scaling and ranking are based on information they collected from web sites and conversations.

TABLE I. IMPORTANT CHARACTERISTICS OF THE MOST PROMINENT OPEN SOURCE SEARCH ENGINES

Platform	programming Language	Supported file formats	Distribution	Scale	Platform support (Out of 5)	Ranking	Users	License
Solr	Java	Many	Yes	TB	5	Very Flexible	Many organizations, e.g. Amazon and Ford.	Apache
Terrier	Java	Many	Yes	TB	5	Very Flexible	Research	Mozilla
Sphinx	C++	Many	Yes	TB	4	Flexible	Craigslist	GPL
Indri	C++	Many	Yes	TB	2	Very Flexible	Research	BSD
Xapian	C++	Many	Yes	TB	3	Flexible	gmane	GPL
Zettair	C	HTML, TREC and TXT	No	TB	1	Flexible	Research	BSD
Katta	Java	Many	Yes	TB	3	NA	Research	Apache
ElasticSearch	Java	Many	Yes	TB	5	Flexible	Many organizations, e.g. Sprint and Grab.	Apache

In this research, we analyze the indexing performance of three frameworks, Solr, Terrier, and Katta. All of these frameworks are chosen due to that all of them are written in Java and can be integrated with Hadoop. Note that, ElasticSearch is also written in java, but Solr and ElasticSearch are both based on Lucene library and have almost similar functionality. So, we choose Solr because it can be fully integrated with Hadoop and ZooKeeper. On the other hand, ElasticSearch is distributed by nature and has its own distributed system, so it does not need to be integrated with Hadoop in order to create processing cluster. Moreover, ElasticSearch is mainly dedicated for JSON file format [49].

Consequently, various frameworks of different indexing strategies have emerged. Each framework has its own specific way of handling and building indexes as well as retrieving documents. Therefore, every framework produces totally different performance in the process of indexing or searching. McCreadie et al. [6] demonstrated that information is increasing every day in every aspect. Thus, Indexing performance for huge collection that contains big number of documents is becoming more challenging. Macdonald et al. [12] stated that indexing such huge collections of documents in single machine requires weeks, if not months, of processing power. Therefore, many researches have been conducted to improve the existing indexing strategies or to propose new indexing strategies that are capable of indexing terabyte-scale data more efficiently, whereas others have produced comparisons of the efficiency between these strategies.

Several researches have proposed an optimization of indexing performance by splitting the process among nodes of cluster. For instance, a new distributed indexing strategy named MapReduce Single-Pass was proposed by McCreadie et al. [2] who also compared the throughput and speedup of the proposed indexing strategy with two other algorithms which include Dean and Ghemawat MapReduce indexing algorithm and Distributed Single-Pass indexing algorithm. Another study by Chen et al. [13] has presented an optimization of Lucene indexing using the distribution technique of MapReduce. Although, they used distributed environment that consists of 4 nodes, they did not study and analyze the scalability and efficiency of their proposed technique. Another study performed by Calvaresi et al. [40] has proposed distributed search system using Apache Hadoop and Lucene. They proposed search system based on real-time request processing that is capable of handling large dataset that exceed the processing capabilities of a traditional computer architecture. Similarly, they did not discuss deeply the scalability of their proposed framework. Velusamy et al. [41] proposed a search engine that employ Hadoop distributed framework to create compressed inverted index data structure in order to be able to handle large amount of data and provide efficient access to stored data. However, the study neither showed results nor analyzed the scalability of their proposed technique.

Other researches have presented comparisons between indexing strategies in terms of indexing performance and searching latency. McCreadie et al. [6] has performed comparison of indexing performance between Terrier indexing, Nutch indexing, Ivory indexing, and MapReduce indexing using multiple TREC datasets and declared that Terrier has provided the most efficient distributed indexing process among the four strategies. Although, the paper focused on comparing the scalability and efficiency of the four strategies, they provided detailed analysis of scalability factors (throughput and speedup) of Terrier only and omit the others. Another comparison study has performed by Turtle et al. [38], which compared the indexing throughput, index size, retrieval effectiveness, and query throughput between Indri and Lucene using data from TREC 6 through 8. Nevertheless, the entire comparison was conducted in a single machine. Moreover, Nagi [14] demonstrated that SolrCloud that uses HDFS as storage system produced the best performance in comparison to the SolrCloud that uses file system of the operating system and also used the LuMongo for distributed real-time search by Lucene. Whereas, the work in [4], and [5] demonstrated the improvement in Solr indexing performance when using MapReduce Hadoop to perform indexing in distributed environment. Similarly, Patel in [48] has proposed system that scale Solr performance using Hadoop. However, the proposed system focused only on the performance of the search functionality and omitted the indexing performance.

Yang et al. [23] expressed that Solr and Terrier are used as the baseline open source search frameworks by many research teams and enterprises, while Katta is the distributed indexing of Lucene search library that is famous and widely adapted open source search framework. Moreover, they are the most widely used open source search engines that employee distributed indexing through the use of Hadoop MapReduce distributed system [39]. Solr, Terrier, and Katta are contemporary indexing platforms. Thus, several researches such as [15], [16], [9], and [17] have utilized Solr system to deploy and investigate their proposed techniques. Additionally, Mutschke et al. [18] chose Solr as basic open source IR framework for evaluating the use of ranking mechanisms and investigating the effect of using different science conceptualizations in the search ranking. Similarly, the works in [19] and [20] have used Terrier to apply and analyze their proposed techniques of TF-IDF models and Probabilistic for retrieving document in IRs. Likewise, Katta has been used in some works such as [42] and [43] that are related to distributed indexing and searching. Other researches such as Ghenai et al. [22] and Benkoussas et al. [21] used the indexing and retrieving functionalities of Solr and Terrier together for applying and examining their proposed studies.

This study differs from previous work by focusing on the on Terrier which has the most efficient indexing strategy as declared by Mccreadie et al. in [6], the most widely used indexing framework (Solr) in business and among researchers, and the distributed indexing of Lucene (Katta) that is widely adapted for open source search libraries [45]. Solr, Terrier, and Katta frameworks are coded using Java, so they can be integrated with Hadoop MapReduce in order to distribute the indexing job across the Hadoop cluster. The comparison between these frameworks aims to provide a detailed overview of their framework architectures and indexing strategies as well as an analysis of the indexing process performance in order to present better understanding of the strengths and weaknesses of these frameworks.

3.0 ARCHITECTURES

In this section, we provide a detailed overview about architectures and main operations for Hadoop, Solr, Terrier and Katta.

3.1 Hadoop Architecture

MapReduce is a paradigm for distributed computing developed for handling a huge amount of data. It is also defined as a programming model, which performs an intensive computation in parallel to process massive dataset by means of splitting the tasks among various processing devices. Google developed MapReduce as a programming paradigm to perform distributed processing for an enormous dataset and to be computed by various tasks as every task carries out equivalent operations on a portion of the defined dataset [37].

Due to the fact that it is free and effective, Hadoop is identified as the best in implementing the programming model of MapReduce. Hadoop MapReduce and Hadoop Distributed File System (HDFS) are considered as the core components of the framework of Hadoop. For more details, HDFS is the Hadoop storage system that enables stream of input and output in a distributed manner to the tasks of MapReduce. Various valuable characteristics are provided by HDFS such as a high level of throughput, fault tolerance and file system with capacity distributed. Moreover, terabytes or petabytes of data cluster storage are provided by HDFS that can be constructed by employing out-of-the shelf equipment. The essential benefits of HDFS in comparison to other distributed file systems are Write-Once, Read-Many and models of stream access. HDFS is ideal for distributing and storing large amounts of data across heterogeneous hardware and operating system environments. Data is split into blocks of specified size, as in the Hadoop the default block size is 64MB. Therefore, this technique allows to duplicate such blocks in the cluster nodes which assists to improve the fault tolerance in data streaming and optimizing the throughput. HDFS was established based on the popular master/slave architecture [24]. In addition, Hadoop MapReduce is a system with a parallel processing capability, which enables each cluster node to carry out equivalent tasks on the portion of dataset entrusted to each one.

Fig. 1 demonstrates Hadoop master-slave architecture in which the master node is called NameNode. According to Taunk et al. [25], the data blocks are distributed by the so called NameNode through the use of mapping table that allows mapping data blocks to DataNodes in order to implement read and write processes from HDFS. Moreover, NameNode is responsible for managing the tasks and controlling namespace of the file system. Likewise it is accountable for client's access to data. On the other hand, the slave node is called DataNode. DataNodes can be either heterogeneous or homogeneous with regard to resources as well as operating system. Data blocks units of DataNodes are stored in which they implement their assigned tasks of processing [26]. Mohamed et al. [8] stated that the ability of data processing locally in the cluster DataNode allows to take the benefits of data locality for improving the cluster DataNode throughput. So, in order to perform a parallel data processing using MapReduce, packaged code, which is required to be processed on each data block, is transmitted from NameNode to DataNodes. Another NameNode are available in case of fault, namely Secondary NameNode that HDFS authorizes to use a copy of metadata stored on the NameNode once there is an error in the NameNode.

Shvachko et al. [24] demonstrated that the DataNode sends a pulse message to the NameNode as a signal of its status and as a request for directions. Thus, DataNodes and clients are enabled to execute the read and write processes via listening to the network when sending the pulse message. Besides, this pulse indicates the presence of the DataNodes, that is to say, if the NameNode receives no response from DataNode within the specified interval, it then declared that DataNode is undetected. Consequently, data blocks stored in such DataNode are considered lost and the NameNode immediately creates new replicas for the blocks of that DataNode.

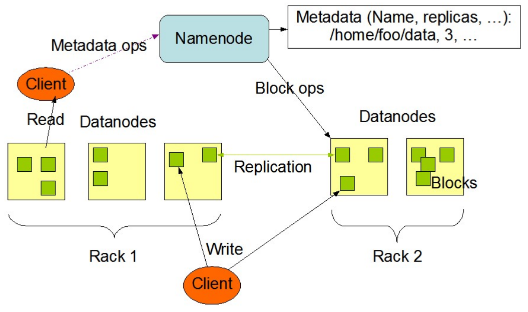


Fig. 1. Architecture of Hadoop HDFS [27].

Fig. 2 shows the typical phases of MapReduce for indexing process. According to Honnutagi and Pooja, [28], there are two phases of MapReduce indexing job namely, map phase and reduce phase. The NameNode splits the dataset into a set of blocks according to the predefined block size. The distribution of the blocks is then performed in a way that ensures the work can be distributed evenly among the available DataNodes. It then specifies the number of map and reduce tasks and distributes them to the DataNodes that have the splits of the dataset. In the map phase, many intensive computation over each dataset split are performed which generates the key/value pairs. As the splits and map tasks are preassigned to each DataNode on the cluster, when a DataNode finishes the current implemented map, it starts a new map tasks and pick up the next splits in its queue. The key/value pairs output of the map task are then sorted alphabetically. Once some map operations finished, the reduce phase, which also can consists of multiple reduce tasks that are distributed among the available cluster nodes, starts to take place for combining and merging the output of each distributed map operation into the required result. In case of DataNode failure, the NameNode distributes the map and reduce tasks of the failed DataNode to the other alive DataNodes.

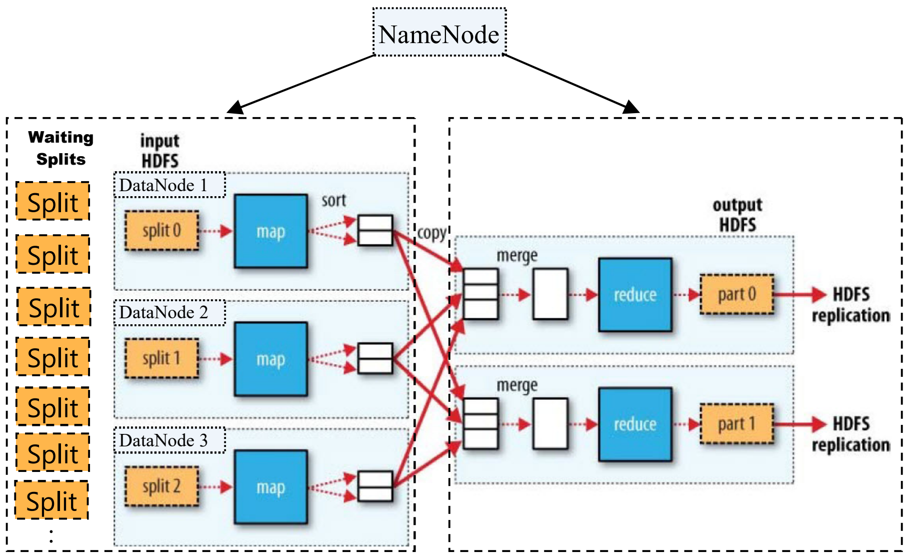


Fig. 2. MapReduce Job Phases [28].

3.2 Solr Architecture

Solr is built on top of Lucene library which is based on inverted index. However, one of the limitations of Lucene is its inability to self-parallelize the indexing process. Hence, in order to overcome Solr indexing issues, Hadoop implementation of MapReduce was used to optimize its indexing process as suggested by Yan et al. [4]. This is done by distributing large number of documents into several shards which enable them to be processed through a cluster consist of various machines. For indexing, Apache Solr is workable in both single-core or multi-core with indexing request handled by an update handler at each core [29]. Like a typical request handler, update handlers are regularly allocated to a particular URL by virtue of employing default or different settings. For each updated handler, Solr creates separate Update Processor Chain (UPC) instance which can execute operations at the document level and it can also transfer the process of indexing to another processing machine.

Common format such as Microsoft Words or PDF, extracted data from database, commas separated value (CSV) files, and XML file also are workable in Solr. Furthermore, working with TIKA enable Solr to operate with various kinds of files beyond its capabilities. When allocating a file to Tika, it spontaneously specifies the file type (e.g., Word, Excel, or PDF, TXT and HTML) in order to extract the text that is needed to be indexed. Moreover, Tika extracts document metadata such as author, title, creating date, etc., as if this is achieved in the presence of schema, they become as a text fields in Solr schema [29].

As illustrated by Yan et al. [4], there are four steps included in the Solr indexing as shown in Fig. 3. The first step is the parser, where text that needs to be indexed from the document is extracted to Lucene workable format, before a document that includes plain text is produced. Then, in second step, the plain text files are packed into object of documents through packaging operation, which creates format-independent abstract document. This process leaves indexing process continuously unaware of the document's original format. The third step involves the tokenization process performed by the tokenizer, where the documents are split into smaller lexical units or tokens. Operation of filtering out tokens is then launched as production and collection of tokens is accomplished in the token stream. Finally, in order to index the data processed by the filter into the index structure, the AddDocument method is called by the IndexWriter. The simplicity of Solr indexing strategy is advocated by Chen et al. [13]. To process the document, tokenization technique is employed throughout the map phase before coming up with another document consist of a list of tokens and their frequencies, that results into the emitting of (Doc-ID, Tokenized-Doc) [53]. The capability to deal separately with each document has been the strong advantage for Solr indexing strategy. Original names of documents are retained when storing newly emitted documents, while original text is indexed and used by same map task. Equally important, since emitting operation is performed by means of each created segment, memory exhaustion is not an issue for this indexing strategy.

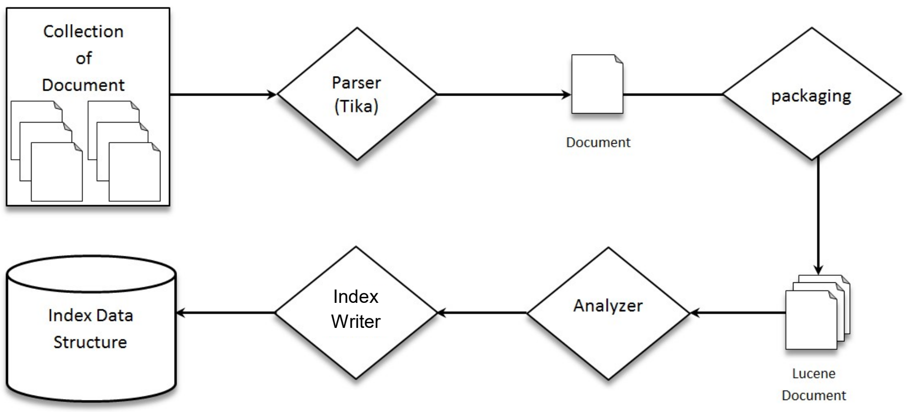


Fig. 3. Indexing process of Solr.

3.3 Terrier Architecture

Single-Pass algorithm is the base of Terrier indexing. Macdonald et al. [12] demonstrated that the main goal of Terrier development is to develop platform that has the ability to scale up with the continuously increasing in dataset size in a centralized environment. Nonetheless, the indexing of centralized single-pass is considered as an extremely slow operation for huge datasets having millions of documents as it needs long time to be accomplished. Therefore, the project then prolonged for supporting the decentralized environment and enables large-scale parallelized indexing utilizing Hadoop MapReduce distribution system. The proposed distribution capability has enabled Terrier to fulfill the requirement of efficient indexing for big collections that consist millions or billions of documents over an appropriate cluster of machines [31].

On other hands, four major data structures are created by Terrier when building index. As illustrated by Ounis et al. [10], these four structures are as follows: document index, direct index, lexicon index, and inverted index. The document index structure includes two essential objects for the document; the length and identifier, while the direct index saves the identifiers terms and frequencies in every document. Similarly, the lexicon index contains the document collection vocabulary and terms frequencies, while inverted index structure becomes the place where produced posting lists are saved. Subject to the application used, Terrier tokenizes and parses every document in the dataset [30]. After that it eliminates stopword and carries out streaming. Via this stage, Terrier generates document and direct indexes, and likewise a part of lexicons is created in the memory to assist decreasing the over-use of memory throughout the indexing processes. As the Document index, Direct index, and Lexicon structures are prepared, they are forwarded to the Inverted File Builder (IFB) that utilizes their data to generate the inverted index structure.

There are many embedded document parsers in Terrier and this allows Terrier to perform indexing for several types of documents, such as HTML documents, plain text, documents MS PowerPoint, Word, and Excel, as well as PDF files. . If Terrier is required to support any file type, the developer should add and make Terrier utilize the plugin of the document type that allows it extract the terms from the document [12]. Terrier can index a collection of documents by means of various methods. Thus, when it is well-configured, it delivers its best performance. Fig. 4 illustrates that Terrier indexing process passes through four stages where plugins can be utilized to support more features and to make changes in the indexing process in each stage. Modality of the structure is considered as essential concerning for the provision of flexibility in each and every step of the indexing process stages which include the extraction of documents from the collection, the extraction of terms from the documents, the tokenization of each extracted document, and the construction of the index structure and writing the index data. Indexing process has many advantages, yet the main one is its ability to index compressed data directly. A corpus will be represented in the form of a Collection object by the indexing process in Terrier, whereas raw text data can be represented in the form of Document object. The indexer, which is part of the TermPipeline chain, is responsible for handling the indexing process as it is divided into many map tasks. Terrier iterates through all the documents collection to extract all the terms to the Termpipeline which tokenizes the terms. The operation in which the pipeline eliminates useless terms is called tokenization. Stopwords and PorterStemmer is a well-known example of Termpipeline chain that uses Stopwords [32].

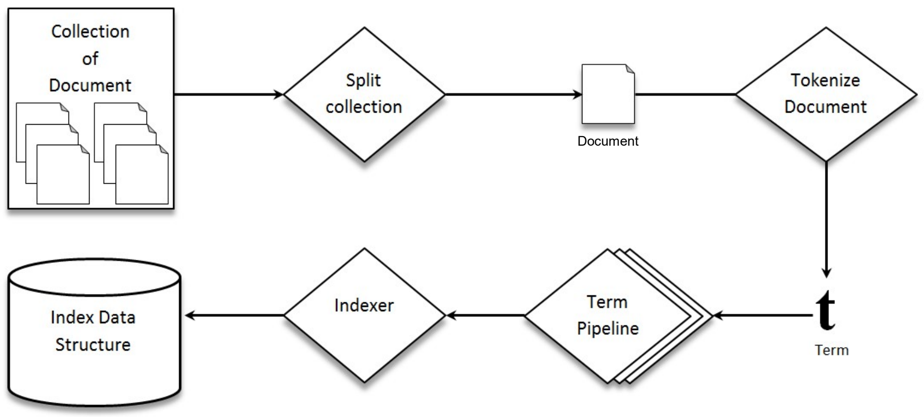


Fig. 4. Indexing process of Terrier.

3.4 Katta Architecture

Although Lucene indexing is one of the most famous indexing platforms, it does not support distributed indexing among multiple servers [4]. Lucene has no fault-tolerance mechanism and it performs indexing in centralized environment (single machine). Therefore, Katta is introduced as an indexing framework that is designed to work on top of Lucene library to add the value of distributed indexing. It is designed to distribute the Lucene indexing process among multiple machines in order to decrease time needed by the indexing process. Thus, Katta is capable of indexing very large volume of data as index shards on multiple machines.

Butler et al. [45] stated that one of the main goals of introducing Katta was the exploitation of many commodity hardware machines similar to Hadoop cluster in the process of indexing. Katta can also be integrated to run with Hadoop cluster. It can exploit Hadoop cluster and distribute the indexing process among all the cluster nodes. Katta is able to read data from HDFS, indexing them using Lucene indexing, then write the indexes back to the HDFS [44]. Furthermore, it replicates shards on different servers for the purpose of improving performance and fault-tolerance.

Katta indexing is a collection of split indexes that are called index shards. Each index shard is a Lucene index that is created using Lucene IndexWriter. Therefore, creating a Katta index is just collecting shards of Lucene indexes together into one folder. Shard Indexes can be zipped in order to reduce the space required to store the index. Katta also merges index with Hadoop in order to provide one unit for searching and deleting of duplicates [46]. According to Calvaresi [40], Lucene has many parsers that are used to extract text data from the documents. One specific parser is invoked at a time according to the type of the document being processed. Katta, distributed Lucene indexing framework, has all the available parsers integrated into Lucene. It has specific parser for PDF files, another parser for DOCX file, and so on. Using these parsers, katta has the ability to extract text from variety of file formats such as PDF files, MS PowerPoint, Word, and Excel documents, as well as HTML and plain text documents.

Katta indexing process is performed throughout three main steps, as shown in Fig. 5. Calvaresi [40] stated that the basic indexing process starts by parsing the document to extract the plain text, then analyzing the tokens that are finally used to build the index structure. Lucene library sequentially reads documents from the collection. Then each document is parsed using the suitable parser. The parsing process is the operation that coverts the input stream into textual representation which is then stored in a file called ContenHandler. ContenHandler is a plain text file that is independent of any file format. The extracted text is then analyzed by a software component called Lucene analyzer used to tokenize the text in the ContenHandler. The output of Lucene analyzer is token list which contains the text words but also can be a URL, an email address, a phrase, or any sequence of characters. In order to achieve better results, the analyzers can perform many additional operations on the tokens such as introducing synonyms and spell check. Finally, the tokens are passed to the Lucene IndexWriter which build the posting list of each token.

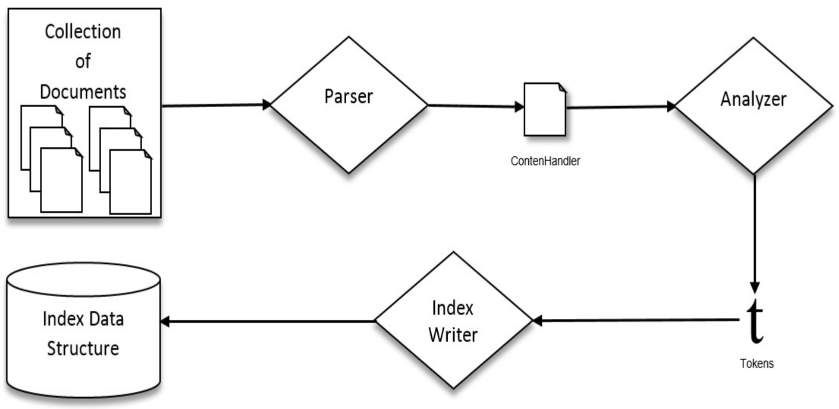


Fig. 5. Indexing process of Katta.

3.5 The differences in the indexing process of Solr, Terrier, and Katta?

The indexing processes for the three indexing strategies seem to be the same as all strategies perform the process of indexing by parsing the documents collection in order to extract the plain texts. Then, the extracted texts are passed through the Tokenizer tool that extracts the Tokens. At that point, the tokens go through a filter which removes the pointless tokens or terms while the rest of tokens or terms are indexed by the indexer of the strategy.

So, what are the differences between the three indexing strategies? From Fig. 3 and 5, it is noticeable that the potential difference between Solr and Katta is that, for Solr and after the Tika process extracts the text content out of the document, it performs the packaging operation that produces an independent file format, while the parser in Katta extracts and creates a plain text file at once. Moreover, the output of the indexing process in the Hadoop cluster is directly written to the Solr, which works simultaneously with Hadoop to store the received index structures in the Hadoop file system (HDFS). Whereas, in the case of Katta, index structure is saved directly to the HDFS. Therefore, Katta requires one more step for deploying the index from HDFS to Katta. In the same line, it is noticed that there are many differences between Lucene (as the base for Solr and Katta) and Terrier. The first difference between them lies in the process of building the index. As inverted index is the base of Lucene indexing, Lucene has a tool called IndexWriter which builds inverted index structures directly. On the other hand, single-pass algorithm is the base of Terrier indexing in which three intermediate structures are prepared by the Indexer as mentioned above. These intermediate structures are then used by the Indexer to build the inverted index. Another important distinction is the technique utilized by each strategy to exploit the local machine memory and how often the index structure is emitted from memory to disk. Emittion is the process of writing the built indexes from the memory to the disk. Unlike Solr, Terrier provide better exploitation of the node memory until it is about to be full. At that point, Terrier emits index structure to the disk. In the contrast, Solr flushes per segment which has number of documents. For Lucene, the IndexWriter starts emittion after indexing a predefined number of documents. It collects and merges the built index structures into one segment then emits the entire segment to the storage disk. Therefore, the emittion in Solr and Katta take place more frequently than that in Terrier. Additionally, there are differences in the execution method between the three frameworks; firstly, Terrier and Katta operate in single instance, while Solr can operate using multiple instances. Secondly, Terrier uses the two phases of the MapReduce job which are the map and reduce, whereas Solr and Katta use only the map phase only. Table II summarizes the most important differences between Solr, Terrier, and Katta.

TABLE II. THE MOST IMPORTANT DIFFERENCES BETWEEN SOLR, TERRIER, AND KATTA

Characteristics	Solr	Katta	Terrier
Emittion (writing indexes from memory to disk)	Per Segment	Per Segment	Memory exhausted
Emittion takes place	More	More	Less
Packaging	Yes	No	No
Map & Reduce Tasks	Map only	Map only	Map and reduce
Instances	More than one	One only	One only
Index structures	Inverted index	Inverted index	Direct index, Document index, Lexicon index, and Inverted index

4.0 EXPERIMENTS AND RESULTS

The experiments aim to evaluate the indexing performance of Solr, Terrier, and Katta. The measurements taken are the average indexing time, the indexing process throughput and speedup as the nodes and dataset size increaseing. Finally, the efficiency is calculated for each experiment as well.

4.1 Experiments Setup

The cluster of the experiments is setup using 10 heterogeneous computers that have GB network adapter and linked together via GB Ethernet switch, Table III gives the detailed specifications of the 10 host computers. The total of 48 cores was divided by two to establish a total of 24 nodes, so that every node has two cores. Oracle Virtual Box version 5.0.26 is used in the experiments to build the virtual cluster of 24 nodes. The operating system of all nodes is Linux Ubuntu 12.04. a compatible and stable version of Hadoop1.0.3[1] is used in all nodes. In addition, HDFS was configured as multi-node cluster to operate in all the 24 DataNodes. There is a node worked as both, the NameNode and DataNode, respectively. The remaining of 23 nodes are DataNodes with 3GB of RAM.

The block size of the HDFS is configured to be 64 MB, so the document collection size is split and stored in blocks of 64 MB. Solr-4.7.2[2] is configured on the NameNode with two Solr instances running in Solrcloud mode that offers automatic load balancing, index replication, failover, and many other specialized features of Solr. Likewise, Terrier 3.5[3] is used in the NameNode. Finally, Katta[4] project is also compiled and implemented on the NameNode.

TABLE III. CLUSTER HOSTS SPECIFICATIONS

No. of Hosts	Processors	No. of Cores	Total Cores	RAM (GB)
2	Intelcore corei7 @3.50GHz	8	16	16
1	Intelcore corei5 @3.20GHz	4	4	8
1	IntelXeon corei5 @3.70GHz	4	4	16
6	IntelXeon corei5 @3.50GHz	4	24	8
Totals cores			48	

4.2 Experiments

Refer to Table III, the cluster was setup to have 48 cores and all cores are allocated for MapReduce jobs. In these experiments, the map tasks number was doubled to 96, so as to maintain all cores busy during task execution and the process of indexing. Moreover, during the operations of the experiments no other tasks were running in the cluster, and the processes of every indexing strategy were repeated three times and then the average of execution time was calculated in order to ensure the accuracy of the results. In these experiments, we used four subsets of standard TERC[5] dataset. The statistics of each subset are presented in Table IV, which includes the subset size in gigabyte, number of documents, average document size in kilobyte, and average document length in words.

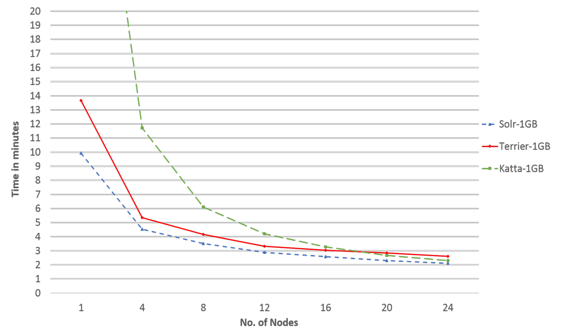
TABLE IV. STATISTICS OF THE 4 TREC SUBSETS USED IN THE EXPERIMENTS

Subset Size	No. of Documents	Average Document Size (KB)	Average Document Length (words/tokens)
1GB	172731	8.2	611
3GB	499881	7.4	563
6GB	1015257	6.6	527
9GB	1593456	6.2	501

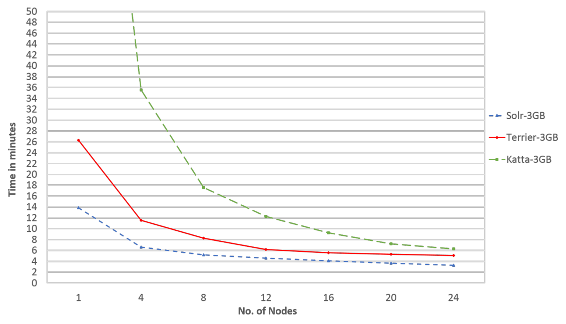
Fig. 6 presents the time spent by the three indexing strategies; Solr, Terrier, and Katta to index a different sizes of datasets. In Fig. 6 (a) and 6 (b), which show the experimental results of using 1 GB and 3 GB datasets respectively, less time was spent by Solr in

[1]http://hadoop.apache.org/
[2]http://lucene.apache.org/solr/
[3]http://Terrier.org.
[4]https://github.com/sgroschupf/katta
[5]http://ir.dcs.gla.ac.uk/test_collections/access_to_data.html

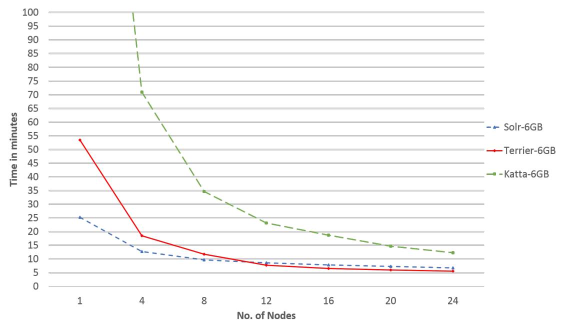
total to complete the process of indexing in various numbers of DataNodes. This is because the nature of Solr indexing is depended on the basis of inverted index, so it immediately builds the inverted index by iterating via the document terms. Once the index structure is constructed for a predefined number of documents, the index in memory is flushed to be stored in the disk. In contrast, a long indexing average time was spent by Terrier due to the fact that there are various intermediate data structures need to be built before establishing the inverted index. Terrier executes this extra operations in the map phase in order to build compressed index structures [10]. Thus the intermediate data structures and their processes make map phase slow. Although, Katta produces the worst indexing time using one node in indexing all datasets (44 mins for 1GB, 133 mins for 3GB, 270 mins for 6GB, and 399 mins for 9GB which do not appear in Fig. 6 due to their big values that are beyond the range of the graphs), its performance becomes better as more nodes are added to the cluster. This means that Katta distributed strategy can provide a good exploitation of available resources and can scale well as more computing resources are added.



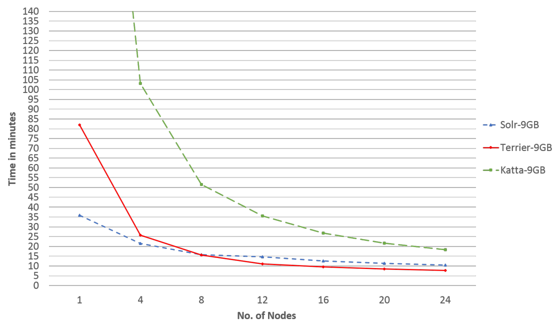
(a) 1GB dataset



(b) 3GB dataset



(c) 6GB dataset



(d) 9GB dataset

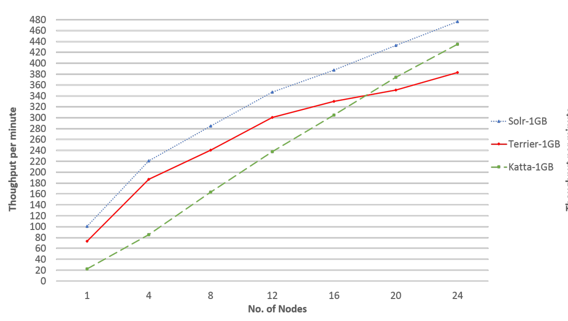
Fig. 6. Average indexing time of Solr, Terrier, and Katta.

As can be seen later in Fig. 8 (a) and 8 (b), we can see that Terrier and Katta produce better speedup than that of Solr in the experiments of 1GB and 3GB datasets. Thus, it is expected that the difference in the indexing performance between the three strategies would change when using bigger datasets. Accordingly, in the 6GB and 9GB datasets experiments as shown in Fig. 6 (c) and 6 (d), Terrier starts to deliver the shortest time of the indexing process when the size of the cluster becomes 12 and 8 nodes for the experiments of 6GB and 9GB datasets respectively. This is because the process of initializing the indexing structures in Terrier take the same time for small and big datasets. Another reason is that the map phase of Terrier indexing produces compressed output which in turn decreases the time taken by the reduce phase. Moreover, it is always said that Terrier is designed from the beginning to scale efficiently with size of the cluster and dataset. Therefore, it always provides better performance as the dataset grows in size. During the execution of the indexing jobs and after some map tasks are performed, Terrier starts also to take advantage of the concurrent execution of the map and reduce tasks. All these reasons play an important role in decreasing the entire time of the indexing process. On the contrary, emittion in Solr takes place more frequently than that of Terrier as Solr emits per segment. In addition, the operations of collecting and merging the index structures of some documents to form the segment takes time. It is also noticed that Solr performs all the operations of the indexing process in the map phase, which prevents it from exploiting the advantages of the concurrent execution of the map and reduce tasks. All of these reasons cause Solr to consume more time than Terrier for indexing bigger datasets.

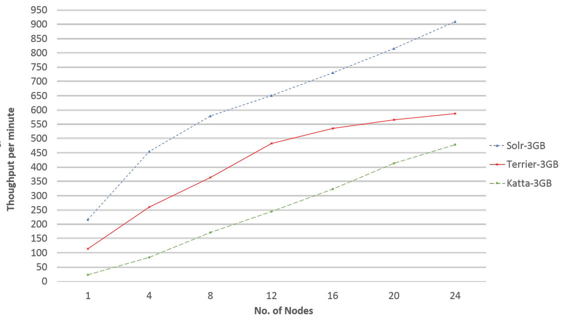
We used Formula (1) to measure the throughput, where *TH* is the generated throughput of the indexing process, m is the number of nodes engaged in the experiment, and T_m is the total time taken to complete the entire indexing process in parallel using $_m$ nodes. The aforementioned formula was used to calculate the throughput in MB/M (megabyte per minute) as the number of nodes is increased. Fig. 7 presents the throughput obtained by each type of indexing strategies in the indexing process of the four datasets as the cluster size is scaled out.

$$TH_m = DatasetSize/T_m \qquad (1)$$

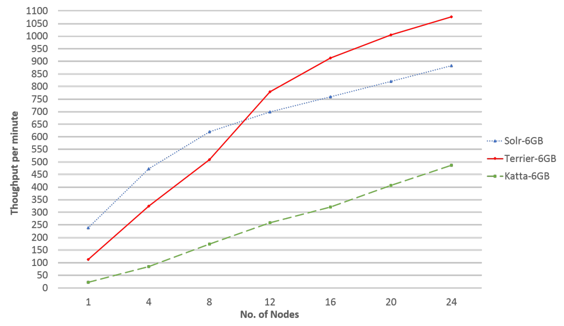
In the experiment of 1GB and 3GB as illustrated by fig. 7 (a) and 7 (b), Solr presented better throughput and it scales linearly as more nodes were added to the cluster. Whilst, the throughput rate of Terrier was reduced as the cluster was scaled out. However, in the experiment of 6GB and 9GB datasets and in the cluster size of 12 and 8 nodes respectively as shown in fig. 7 (c) and 7 (d), Terrier indexing starts to deliver better throughput and it increases in a linear fashion as the number of nodes allocated for the indexing is increased. On the other hand, Solr performance was reduced at the same points of the cluster sizes due to the lengthy map tasks which compromised its linear scaling.



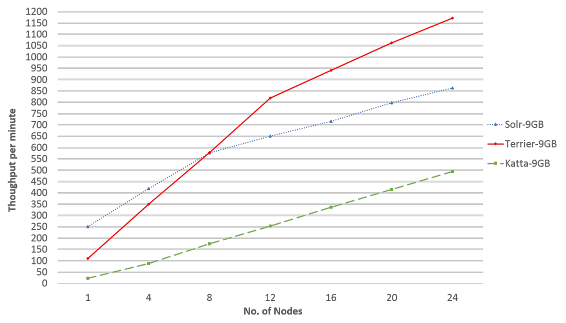
(a) 1GB dataset



(b) 3GB dataset



(c) 6GB dataset



(d) 9GB dataset

Fig. 7. Achieved throughput of Solr, Terrier, and Katta.

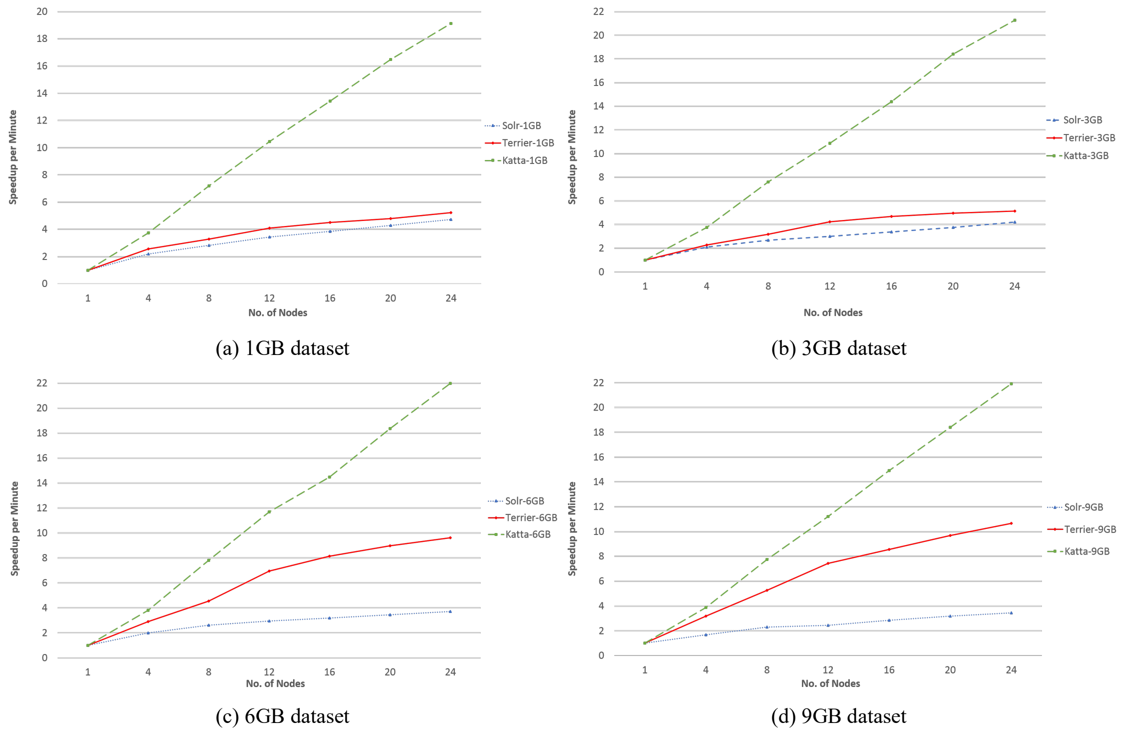
Formula 2 is used to measure the speedup, where S_m represents each indexing process speedup, T_i is the entire time needed to index the same dataset in one machine. Fig. 8 shows that the achieved speedup of the three strategies increases as the nodes number and dataset size are increased. Solr achieves the best indexing time in the 1GB and 3GB experiments as depicted in Fig 6 (a) and 6 (c). However, Fig. 8 (a) and 8 (b), which represent the indexing speedup of 1GB and 3GB datasets, show that the speedup of Katta and Terrier is better than that of Solr. This superiority of Katta and Terrier is due to the good exploitation of the available resources. Although, this good exploitation of available resources in Katta is better than that of Terrier, but Terrier produces better indexing time when indexing bigger datasets as shown in Fig. 6 (c) and 6 (d). It is because that the Terrier map task exploits the advantages of increasing number of the cluster nodes and the bigger size of the dataset to scale in a way that is suitable for the indexing process. What was not expected is that Katta keeps producing the longest indexing times, however it produces the best speedup among the three. This is probably due to that the datasets are still not large enough to show the reflection of the good speedup. It is also may be that its distributed algorithm takes more time to distribute data and tasks, and to collect results from cluster nodes. Furthermore, Katta performs the process of emittion more frequently than Terrier. Katta's emittion takes place for every segment which also needs time to be gathered from all nodes and merged before it can be written to the disk. Moreover, Solr and Katta perform all the operations of the indexing process in the map phase, this prevents them from taking the advantages of the concurrent execution of the map and reduce tasks. All the above mentioned reasons cause Solr and Katta to need more time than Terrier to finish the 6GB and 9GB datasets collections.

$$S_m = T_i / T_m \qquad (2)$$

Finally, this study focuses on calculating and analyzing the efficiency of each indexing strategy. The Efficiency (E) is a measure of each indexing strategy for the fraction of time in which a processor is usefully employed. In practice, efficiency value is always between zero and one, according to the degree of effectiveness on how the processors of cluster nodes are exploited. Since efficiency is the ratio of execution time in single node to execution time in multiple nodes, the optimal distributed system has an efficiency of one. Therefore, the efficiency of each indexing strategy is calculated using Formula 3.

$$E = S_m / i \qquad (3)$$

Where E is the efficiency of each indexing strategy, S_m is the speedup of the same indexing strategy for each experiment, and i is the number of processors used in each indexing experiment. Fig. 9 depicts the efficiency achieved by each strategy in the indexing process of the four datasets as the cluster size is scaled out. It is clear that in all experiments, as shown in all subfigures, the value of the efficiency start to decrease as the number of nodes is increased. However, when the size of the dataset is increased, the efficiency increases too. This is due to that parallel computing is meant for large data set. Thus, a larger dataset size yields higher efficiency in exploiting the available number of cluster nodes. Although, Katta has achieved the worst average indexing time as showed in Fig. 7, it produced the best exploitation of the available cluster nodes. Efficiency reflected by speedup, Katta shows almost to linear speedup in Fig. 8 and as a result the efficiency shows in Fig. 9 is almost to 1.



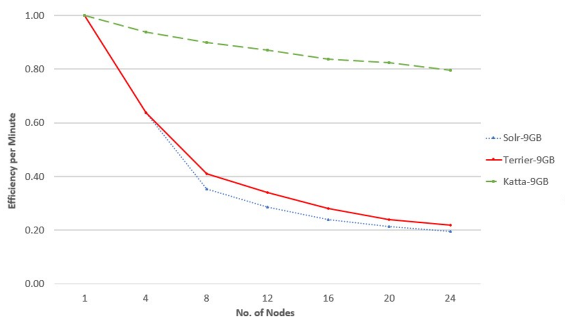
(a) 1GB dataset

(b) 3GB dataset

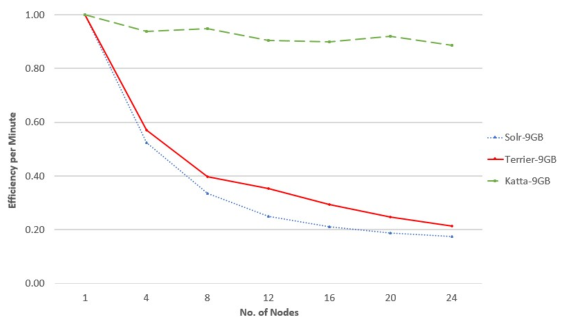
(c) 6GB dataset

(d) 9GB dataset

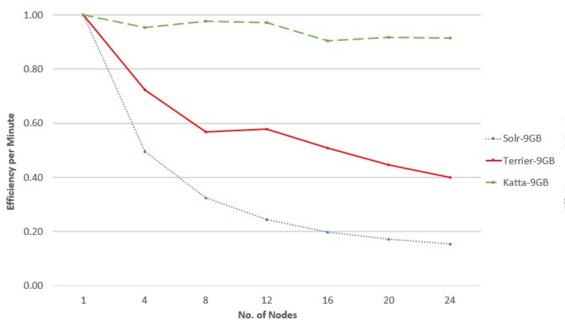
Fig. 8. Achieved speedup of Solr, Terrier, and Katta.



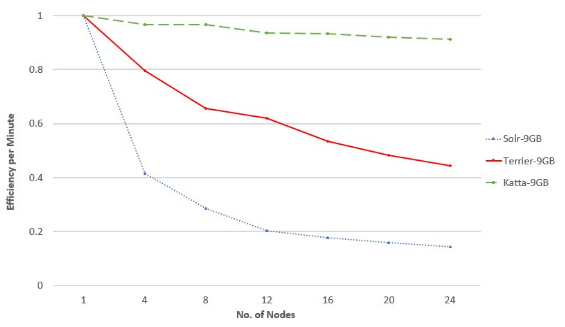
(a) 1GB dataset



(b) 3GB dataset



(c) 6GB dataset



(d) 9GB dataset

Fig. 9. Achieved efficiency of Solr, Terrier, and Katta.

Based from the experiments, it is obvious that Terrier has the capability to deal with huge data sets more efficiently as it achieved the shortest average indexing time with bigger datasets. On the contrary, Solr is recommended to use when there is limited computation power by using the technique of continuous feeding of small size documents. Note that, Katta has achieved the longest average indexing time with all datasets. However, Katta has achieved the best speedup among the three indexing frameworks, which means Katta is good and efficient for indexing very huge terabyte-scale datasets. In order to summarize the experiments results, Table V gives a summary of the experiments measurement results and analysis of the four (4) performance metrics. The right sign (✓) shows that, the indexing frameworks are the best for that dataset and performance metrics, while the dash sign (-) shows that, the indexing framework is not performing well for that dataset and performance metrics.

TABLE V. SUMMARY OF THE EXPERIMENTS MEASUREMENT RESULTS AND ANALYSIS OF THE FOUR (4) PERFORMANCE METRICS.

dataset	Solr				Terrier				Katta			
	Indexing Time	Throughput	Speedup	Efficiency	Indexing Time	Throughput	Speedup	Efficiency	Indexing Time	Throughput	Speedup	Efficiency
1GB	✓	✓	-	-	-	-	-	-	-	-	✓	✓
3GB	✓	✓	-	-	-	-	-	-	-	-	✓	✓
6GB	-	-	-	-	✓	✓	-	-	-	-	✓	✓
9GB	-	-	-	-	✓	✓	-	-	-	-	✓	✓

5.0 CONCLUSION

This study investigates the distributed indexing of three famous information retrievals frameworks Solr, Terrier and Katta, over the use of MapReduce programming model. In particular, the performance of Solr, Terrier and Katta distributed indexing is investigated in terms of indexing average time, throughput, speedup as well as efficiency. The experiments are conducted by analyzing the indexing performance on multiple machines (cluster) and using different TREC dataset sizes. Interestingly, Terrier produces the best performance result with large collection and also produced better exploitation of available distributed resources when the collection size increases. This is because Terrier always exploits the memory of the local node and also uses compression techniques for reducing the output of map phase. On the contrary, Solr proves to have an efficient indexing performance with small collections. This is a result of building directly the inverted index structure without preparing an intermediate structure. However, with big dataset such as 6GB and 9GB, Solr produces slower performance than Terrier. For Katta, the experiments of one node indexing deliver the longest average indexing time for all datasets. However, its performance became better as the cluster scaled out and larger dataset was used. Thus, the results showed that Katta utilizes the added resources more effectively. Due to this, Katta achieved the best speedup and efficiency among the three frameworks. As indicated by the analysis of the experimental results, Terrier is preferred in the case of indexing large collections in the presence of scalable computing power. Whereas, Solr is recommended when having small collections and limited processing power. For Katta, it is recommended to be examined using larger datasets in order to analyze whether the good speedup can benefit the indexing average time for larger dataset. The results in this study is a great benefits for researchers and practitioners to understand the difference between the three distributed indexing framework and assist them to choose the appropriate indexing framework. The evaluation of distributed query execution is our future plan. The experiments will investigate the query latency and query throughput.

ACKNOWLEDGEMENTS

This research is fully funded by the Universiti Putra Malaysia under the Grant Putra, GP/2017/9569600.

REFERENCES

[1] A. Y. Aldailamy, N. A. Wati Abdul Hamid and M. Abdulkarem, "Performance Evaluation of Distributed Indexing Using Solr and Terrier Information Retrievals," *2018 Fourth International Conference on Information Retrieval and Knowledge Management (CAMP)*, Kota Kinabalu, 2018, pp. 1-5. doi: 10.1109/INFRKM.2018.8464814

[2] R. Mccreadie, C. Macdonald, and I. Ounis, “Comparing distributed indexing: To MapReduce or not?,” in *LSDS-IR’09 7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, July 23, 2009, Boston, MA, USA: ACM SIGIR 2009.

[3] P. Mika, “Distributed indexing for semantic search,” in *SEMSEARCH 10 3rd International Semantic Search Workshop,* April 26 - 26, 2010, Raleigh, North Carolina, USA: ACM New York, NY, USA 2010.

[4] L. Yan, S. G. Liu, and D. X. Lao, “Solr Index Optimization Based on MapReduce,” *Applied Mechanics and Materials*, vol. 556-562, pp. 3506–3509, May 2014.

[5] N. B. Hung, ”An Implementation of Vector Space Model Using Hadoop Platform for Private Document Retrieval Systems,” *Journal of Science and Technology - Technical University*, Vol. 97, pp. 63-69, Jan. 2013.

[6] R. Mccreadie, C. Macdonald, and I. Ounis, “MapReduce indexing strategies: Studying scalability and efficiency,” *Information Processing & Management*, vol. 48, no. 5, pp. 873–888, Sep. 2012.

[7] R. Blanco, P. Mika, and S. Vigna, “Effective and Efficient Entity Search in RDF Data,” in *The Semantic Web – ISWC 2011, Lecture Notes in Computer Science*, vol 7031, pp. 83–97, 2011, Springer, Berlin, Heidelberg.

[8] H. Mohamed and S. Marchand-Maillet, “MRO-MPI: MapReduce overlapping using MPI and an optimized data exchange policy,” *Parallel Computing*, vol. 39, no. 12, pp. 851–866, Dec. 2013.

[9] A. Atreya V, S. Chaudhari, P. Bhattacharyya, and G. Ramakrishnan, “Building Multilingual Search Index using open source framework,” in *SANLP 3rd Workshop on South and Southeast Asian Natural Language Processing,* Dec. 8, 2012, Mumbai, India: COLING 2012.

[10] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma, “Terrier: A high performance and scalable information retrieval platform,” in *second workshop on Open Source Information Retrieval (OSIR),* August 10, 2006, Seattle, WA, USA: ACM SIGIR 2006.

[11] C. Middleton and R. Baeza-Yates, “A comparison of open source search engines,” *Universitat Pompeu Fabra,* Barcelona, Spain, Nov. 2008.

[12] C. Macdonald, R. McCreadie, R. L.T. Santos, and I. Ounis, “From puppy to maturity: Experiences in developing Terrier,” in *Workshop on Open Source Information Retrieval,* August 16, 2012, Portland, Oregon, USA: ACM SIGIR 2012.

[13] Z. Chen, C Zhu, W Cheng, Q. Song, and S. Cai, “Research of Distributed Index Based on Lucene,” *Advances in Electronic Engineering, Communication and Management,* Vol.1, pp. 115-121, Jan. 2012, *Lecture Notes in Electrical Engineering,* vol 139, Springer, Berlin, Heidelberg.

[14] K. Nagi, “Bringing Search Engines to the Cloud using Open Source Components,” *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management,* 2015.

[15] L. Ma, W. Bao, W. Bao, W. Yuan, T. Huang, and X. Zhao, “A Mongolian Information Retrieval System Based on Solr,” *2017 9th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, Jan.14-15, 2017.

[16] J. Woo, “Information Retrieval Architecture for Heterogeneous Big Data on Situation Awareness,” *International Journal of Advanced Science and Technology*, vol. 59, pp. 113–122, Oct. 2013.

[17] H. Zhao and X. Chen, “Chinese Tourism Information Search Platform based on Cloud Computing,” *Proceedings of the 2015 International Industrial Informatics and Computer Engineering Conference*, 2015, Beijing, China.

[18] P. Mutschke and P. Mayr, “Science models for search: a study on combining scholarly information retrieval and scientometrics,” *Scientometrics*, vol. 102, no. 3, pp. 2323–2345, 2014.

[19] A. Tamrakar, and S. Vishwakarma, ”Analysis of Probabilistic Model for Document Retrieval in Information Retrieval,” 2*015 International Conference, Computational Intelligence and Communication Networks (CICN),* 2015.

[20] A. Mishra and S. Vishwakarma, “Analysis of TF-IDF Model and its Variant for Document Retrieval,” *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, Dec. 12-14 2015 Jabalpur, India.

[21] C. Benkoussas and P. Bellot, “Cross-Document Search Engine for Book Recommendation.” Proceeding in the *CBRecSys@ RecSys,* Sep. 20, 2015, Vienna, Austria.

[22] A. Ghenai, E. Khalilov, P. Valov, and C. L. A. Clarke, “WaterlooClarke: TREC 2015 Clinical Decision Support Track,” *University of Waterloo,* Canada, 2015.

[23] G. H. Yang and I. Soboroff, “TREC 2016 Dynamic Domain Track Overview,” *Proceeding in the TREC 2016,* Nov. 15-18, 2016, Gaithersburg, Md. USA.

[24] K. Shvachko, H. Kuang, and S. Radia, “The Hadoop distributed file system.” 2010 IEEE 26th symposium on Mass storage systems and technologies (MSST), May 3-7, 2010, Lake Tahoe, Nevada, USA.

[25] A. K. TAUNK, A. K. Parmar, and R. Srivastav, “The Hadoop Distributed File System,” *International Journal of Computer (IJC),* vol. 8, no. 1, pp. 8-15, 2013.

[26] C. He, D. Weitzel, D. Swanson, and Y. Lu, “HOG: Distributed Hadoop MapReduce on the Grid,” *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov. 10, 2012.

[27] M. Abdulkaremand R. Latip. "Data Transmission Performance Analysis in Cloud and Grid," *ARPN Journal of Engineering and Applied Sciences,* VOL. 10, NO. 18, pp. 8451- 8457, Oct. 2015.

[28] P. S. Honnutagi, "The Hadoop distributed file system," *International Journal of Computer Science and Information Technologies (IJCSIT),* Vol. 5, no. 5, pp. 6238-6243, 2014.

[29] H. V. Karambelkar, "Scaling Big Data with Hadoop and Solr," 2nd ed. Reading: Packt Publishing Ltd, 2015, [E-book] Available: Google e-book.

[30] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and D. Johnson, "Terrier Information Retrieval Platform," *Lecture Notes in Computer Science Advances in Information Retrieval*, pp. 517–519, 2005.

[31] A. Couste, J. Kozowski, and W. Martin, "A Performance Analysis of Distributed Indexing using Terrier," *StuConOS 2012 University College London*, Nov. 26, 2012, United Kingdom, RN Journal, vol.12, 2012

[32] P. Goswami, "Batch (TREC) Indexing and Retrieval using Terrier 2.2." *Indian Statistical Institute*, Kolkata.

[33] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft, "Indri: A language model-based search engine for complex queries," *Proceedings of the International Conference on Intelligent Analysis,* Vol. 2, No. 6, may 2-4, 2005.

[34] R. C. Miller and K. Bharat, "SPHINX: a framework for creating personal, site-specific Web crawlers," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 119–130, 1998.

[35] J. Zobel, H. Williams, F. Scholer, J. Yiannis, and S. Hein, "The zettair search engine," *Search Engine Group,* RMIT University, Melbourne, Australia, 2004.

[36] "The Xapian Project," an Open Source Probabilistic Information Retrieval Library. [Online]. Available: http://www.xapian.org/. [Accessed: 10-July-2017].

[37] J. Dean and S. Ghemawat. "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[38] H. Turtle, Y. Hegde, and S. Rowe, "Yet another comparison of Lucene and Indri performance," *SIGIR 2012 Workshop on Open Source Information Retrieval,* Portland, Oregon, USA, Aug. 16, 2012.

[39] W. B. Croft, D. Metzler, and T. Strohman, "Search engines: Information retrieval in practice," Vol. 283. Reading: Addison-Wesley, 2015, [E-book] Available: Wiley Online Library.

[40] M. Calvaresi, "Building a Distributed Search System with Apache Hadoop and Lucene." *Universita di Roma* ,2012.

[41] K. Velusamy, D. Venkitaramanan, N. Vijayaraju, G. Suresh, and D. Madhu,. Inverted indexing in big data using hadoop multiple node cluster. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *4*(11), 2013.

[42] A. Singh, S. Kulkarni, S. Banerjee, G. Ramakrishnan, and Chakrabarti, S, "Curating and searching the annotated web" . In *SIGKDD Conference*, 2009.

[43] D. Diaz-Sanchez, F. Almenarez, A. Marin, P. Arias, R. Sanchez-Guerrero, and F. Sanvido, "A privacy aware media gateway for connecting private multimedia clouds to limited devices." In *Wireless and mobile networking conference (WMNC), 2011 4th joint IFIP* (pp. 1-8). IEEE, , Oct. 2011.

[44] H. B., Mark and J. Rutherford, "Distributed Lucene: A distributed free text index for Hadoop." *HP Laboratories*, 2008.

[45] T. Lee, H. Lee, K. H. Rhee, and U. S. Shin, "The efficient implementation of distributed indexing with Hadoop for digital investigations on Big Data." *Computer Science and Information Systems*, *11*(3), 1037-1054, 2014.

[46] Katta, Lucene & more in the cloud, http://katta.sourceforge.net/

[47] C. Gormley, and T. Zachary, "*Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*." O'Reilly Media, Inc., 2015.

[48] T. Patel, D. Patel, R. Patel, S. Shah, "Scaling Solr Performance Using Hadoop for Big Data." (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 8 (1), 31-33, 2017

[49] M. S. Divya, , and S. K. Goyal. "ElasticSearch: An advanced and quick search technique to handle voluminous data." *Compusoft* 2, no. 6, 171, 2013.

[50] Jeff's Search Engine Caffè, "Information Retrieval research and search engine development discussion." http://www.searchenginecaffe.com/2007/03/open-source-search-engines-in-java-and.html

[51] J. Felt, Open Source Search Comparison, https://gist.github.com/jeremyfelt/8230088

[52] V. Singh, "A Comparison of Open Source Search Engines," https://partyondata.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/

[53] Butler, Mark H., and James Rutherford. "Distributed Lucene: A distributed free text index for Hadoop." *HP Laboratories* (2008).

[54] T. Grainger, T. Potter, and Y. Seeley, " *Solr in action*." Cherry Hill: Manning, 2014.

[55] W. B. Frakes, and R. Baeza-Yates, eds. *Information retrieval: Data structures & algorithms*. Vol. 331. Englewood Cliffs, NJ: prentice Hall, 1992.

[56] I. Biswas, and V. Phadke. "Project Report Comparative Analysis of Data Structures for Inverted File Indexing in Web Search Engines." *Google Scholar*.