ISSN 2320-2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

Indexing Strategies of MapReduce for Information Retrieval in Big Data

Mazen Farid, Rohaya Latip, Masnida Hussin, Mohammed Abdulkarem
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, (UPM), Serdang, Selangor, Malaysia
mazenfareed7@yahoo.com, rohayalt@upm.edu.my, masnida@upm.edu.my, memo.ye@gmail.com

Abstract *:* In Information Retrieval (IR), the efficient strategy of indexing large dataset and terabyte-scale data is still an issue because of information overload as the result of increasing the knowledge, increasing the number of different media, increasing the number of platforms, and increasing the interoperability of platforms. Across multiple processing machines, MapReduce has been suggested as a suitable platform that use for distributing the intensive data operations. In this project, sensei and Per-posting list indexing (Terrier) will be analyze as they are the two efficient MapReduce indexing strategies. The two indexing will be implemented in an existing framework of IR, and an experiment will be performed by using the Hadoop for MapReducing with the same large dataset. In particular, this paper will study the effectiveness of two indexing strategies (Sensei & Terrier), and try to find and verify the better efficient strategy between them. The experiment will measure the performance of retrieving when the size and processing power enlarge. The experiment examines how the indexing strategies scaled and work with large size of dataset and distributed number of machines. The throughput will be measured by using MB/S (Megabyte per Second), and the experiment results analyzing the performance and efficiency of indexing strategies between Sensei & Per-posting list indexing (Terrier).

Keywords*:* Hadoop, Indexing, MapReduce, Sensei, Terrier.

INTRODUCTION

The Web is the large place to store documents, and has the main challenge for Information Retrieval systems (IR), which used by search engines of the Web or Web IR researchers. Index should be created in order to increase the efficiency of documents information retrieval. Index is considered the special data structure which used in this regard. The dataset usually contains many documents which are stored in one hard disk and sometimes in more than one hard disk. The indexing, therefore, should cover different or many hard disks in which the documents are stored. The IR system works through inverted index in which every term has a posting list. This posting list represents the documents through numbers or integer documents (IDs), and they also contain the terms as stated by [1]. Every document has a representing score which stored in the posting list. The importance of this process is to figure out the information sufficient statistics. The main goal here is to find the sufficient statistics of information, e.g. the terms frequencies which are occurred, and the information of position

The textual indexes usually stored with lexicon which are considered additional structures. These lexicons include pointers which are important for the posting list that added to the inverted index. In the final stage the result of the search can be shown through using the information of the documents such as its name and its length. This is used to display the documents in a specific order for the user. This is what is called indexing and an important point is that it should be in the mode of offline before the process of searching is done.

The single pass indexing used in Terrier system that released with Terrier 2.0 [2], this term is used to describe the idea of building the document of single pass central structure over the collection. In addition, single pass use low memory consumption when creating the index. This process is achieved through compressing the inverted files as well as creating a temporary posting list that used in one single machine only. According to memory consumption, this type of indexing is considered the most efficient and the faster.

In distributed systems is that Terrier supports huge dataset indexing through using the functions of Hadoop's MapReduce by using a single pass indexer. There are three organizations for the output of functions of the Map. The first organization is information saved about the document during its run. The second organization is the indices of each document for every map task. The third organization is the list of the term as well as its posting list.

When the space of the memory and power of processing is limited, the sharding strategy can be used. Sensei used this strategy through the creation of the fast MapReduce job. This is achieved through taking utilizing the data from Hadoop with the given schema. Sensei cannot support the JOINS, but it can create a single index. Therefore, the aim of this study is to explain and clarify the advantages of the process of indexing large datasets through utilizing MapReduce.

ARCHITECTURE

In this section, we give some brief description of materials and architectures that covered in this paper.

Terrier architecture

Terrier supports many different ways to indexing the data of documents. As shown in Fig. 1, there are four stages of the process of indexing. The indexing process at each stage can be

ISSN 2320-2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

changed by its entities [3]. According to this architecture, there is flexibility in the various stages of the process of indexing which are the documents dataset handling, the parsing of each individual document in both the process of terming the documents and the writing of the structures of the index data. Moreover, the main and most important advantage of Terrier is related to compressed data through the allowance of the direct express and indexing.

Terrier can index many types of documents such as HTML, Plain text documents, Excel, Ms Word, PowerPoint, and PDF files through having their embedded parsers. The developers can add other types of documents and index them. This can be achieved through adding the formats plugins of the files so that the terms can be extracted from them.

There are three main properties for each grabbed term: the string of the term textual form, the term occurrence position in the document, and the field where the occurrence happens [3].

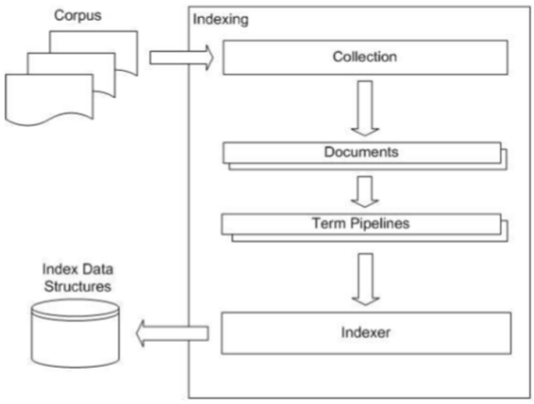

Fig. 1: Indexing architecture of Terrier.

The terms are transformed in different ways during the "Pipeline term", and the role of Terrier in this stage is to add aliasing.

Algorithm and stopword removal components are the two predefined variants plugins of porters. The flexibility term processing is one of the characteristics of Terrier. Indexing is the considered that last step in pipelining. The indexer has the responsibility to create the index of appropriate data to create new data structure called indexing. The number of the blocks determines the division of the documents which in turn considered the factor that determines the accuracy of the terms registered. There is an index in each block created by the block indexer. This index stores each term position in the document blocks.

Terrier is based on the process of single-pass indexing. One can see that the indexing process can be split into the number of tasks for mapping the documents [1]. Every map task in the process can operate on its own subset. The term posting list compression in the memory is achieved through the compression process. The all documents which are processed in the memory run very low, and the partial index is erased completely from the mapping task, this process can be done by emitting the pair of <term, posting list> for each term. The indices which are erased are sorted, map and flushed numbers before they are passed into a reduced task. The posting list for every term is created by the reduce function which collect their flushes and merge them. The map numbers and flush number are used to make sure of the posting list correct order. The standard indexing is created when the posting list of the term is taken by the reduce function and post it in the final posting.

Sensei architecture

There is a need for sharding indexing when there is a huge bulk of data and it cannot be handled by the single machine [11]. Also, when the power of the CPU is limited or there is not enough space in the disk, the sharding indexing is used. The Sensei is designed for the full text search of LinkdIn as it utilized the sharding indexing the huge data the contains the SQL-variant query language which is called Browse Query Language (BQL), JSON Query format in order to index them in certain format of data < key, value>. Sensei used Hadoop for support the multi node layers and using its MapReduce functions. Also, Sensei can deal with the parameters numbers which are used to determine the number of the shards and the division of the shards of the document. Sensei also gives number of plugins in order to allow the manager of indexing to specify the data belongs to shards. Fig. 2 shows the indexing architecture of Sensei.

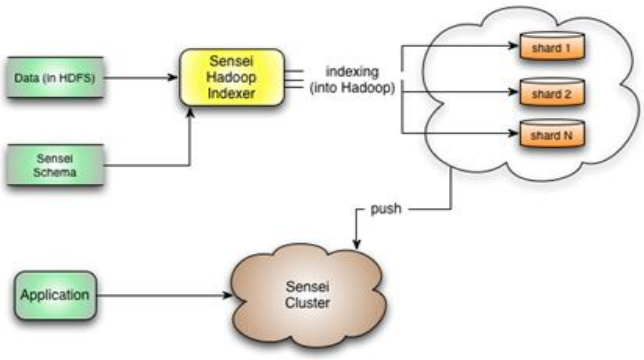

Fig. 2: Indexing architecture of Sensei.

Hadoop architecture

There are two main components of the cluster of Hadoop which are: Hadoop Distributed File System (HDFS) and MapReduce [4]. HDFS in Hadoop cluster are important to deal with its data [5]. HDFS gives the storage requirement for MapReduce data input and output data for many reasons: (1) It is devised as an exceptionally fault tolerant, (2) high throughput, (3) high capacity distributed file system. It has the ability of saving large cluster data of terabytes or petabytes [5]. Also, it has various hardware requirements which has commercial hard ware just like PCs. There are many differences between HDFS and the other distributed systems: (1) the task nature of HDFS is to read-many and write-once, (2) the stream model here gives the ability to HDFS to distribute the data efficiently, (3) the storage of a huge bulk of

ISSN 2320-2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

dataset, (4) and the join of heterogeneous operating systems as well as hardware environments. In addition, there is a division of each file to the number of block size of (64 MB) and storage of the duplicates in each disk of cluster nodes. The mass dataset increases the block numbers that increases the tolerance of the fault. HDFS is efficiently used in the architectures that utilize master/slave.

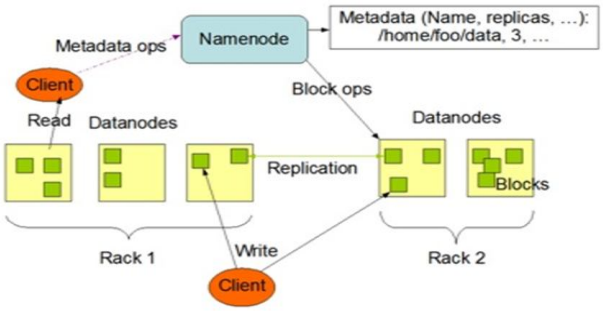

Fig. 3: HDFS architecture.

The master known as NameNode [4], which manages customer's access to data and deal with file system namespace. There are several numbers of operating nodes called DataNodes, that can store the real data in form of certain blocks. The NameNode build the mapping table to maps the data and blocks to DataNodes to manage the write and read orders in the HDFS clients.

Furthermore, HDFS gives the permission to copy the metadata in secondary NameNode when the error occurs in the master NameNode. The DataNodes store the blocks of data in their local disks and perform commands, for example copying, creating, deleting and substituting data. HDFS is accountable for file system operations of namespace, such as opening, closing, renaming files and directories. Fig. 3 [4] shows the architecture of HDFS.

The status of DataNode reported according to the requests and pulse messages of NameNode for directions. The ability of listening to each DataNode in network accordingly lets the other DataNodes and clients to perform write and read operations. Furthermore, the pulse helps to identify the communication between NameNode and its DataNodes. In case the DataNode does not receive the pulse from the NameNode at the appropriate period of time, data blocks which stored at that DataNode are indicated as lost blocks and the NameNode copy the lost blocks of that DataNode [8].

METHODOLOGY

In the study of the indexing strategies (Sensei & Terrier), both of them use Hadoop's MapReduce. MapReduce executed over multiple machines where the data typically not store in the central file's store, and it distributed into blocks (64 MB) across many machines [1]. This mean the map functions have the ability to operate on data that can be locate in local machines without needing of transit intra or intra-data center backbone links, and prevent overload of the central file storage service. Therefore high bandwidth can be achieved by local CPUs because data is always as local as possible, and the Intermediate results of map tasks are stored on the processing machines themselves [9].

Good throughput thus can be performed due to the permanent locality of data to the processor which performs the function. The results of map jobs, that are intermediary, are stored at the local machine itself. Reduced functions at each machine decrease the volume of output and merge it by utilizing some combiners. A central machine provides job and task's scheduling, which tries to make jobs as local as possible to input data. A central machine provides job and task's scheduling, which tries to make jobs as local as possible to input data [9].

The main job here focuses on which of these two strategies can index the large size of dataset quickly and with more efficiency in term of speed-up.

In our hypothesis Terrier is the best indexing strategy comparing with the other strategies. Using Terrier in searching machines enhanced the searching performance by using a single-pass indexing method [6], the compressed posting lists for each term are built in memory as the dataset is scanned.

However, it is not preferred to store the posting lists of many documents in single machine's memory [1]. When the memory exhausted the partial indices are flushed to disk, and the final index created by merging the partial flushed indices.

To implement the experiment, the following systems were installed:

Terrier-3.5

Terrier applies the art of indexing and retrieval tasks [10], and provides an optimum platform for fast evolution and development of retrieval applications for large dataset.

Sensei-1.5.0

Sensei used Hadoop for support the multi node layers and using its MapReduce functions [11]. Sensei deal with the number of parameters used to determine how many and what shards the document can be divided to.

Hadoop-1.0.3

Hadoop[7], implements MapReduce functions with two main categories of component Job-tracker and number of Task-trackers. Hadoop defines how Jop-tracker commands the Task-trackers to deal with data in parallel through two main tasks Map and Reduce.

EXPERIMENT

The result of the underlying experiments will test the data size and time in seconds, and the result will be measured in terms of MB/S (Megabyte per Second). The experiment aims and objectives are:

- Examine the two strategies with the same dataset within different number of machines.

ISSN 2320-2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

- Sensei vs. Per-posting list indexing (Terrier).
- Measuring the speed-up, $S_m = T_1/T_m$, where *m* is the number of machines, T_1 is the execution of the algorithm on a single machine, and T_m is the execution time in parallel, using m machines [1].

In this section we offer details of our experimental setup, as it consists three hosts linked through the D-link WAN routers. The hosts fit out with the Intel series of processors with i5 processor operating at 1.8GHz with 8GB RAM, 10/100MB network adapter, and windows 7 operating system. Oracle Virtual box 4.1.18 r78361 is utilized for Linux Ubuntu release 10.04 operating system [8], 10 virtual machines with 1GB RAM , 113.5 MB dataset with 360000 records [8].

Installing Terrier doesn't take a long time and it is easy to configure with Hadoop for multi-node, taking the advantages of Hadoop features for MapReducing and Hadoop distributed file system (HDFS).

Terrier supports the concept of desktop search engine; according to increase the requirement of information retrieval, Terrier can easily apply the suitable application for retrieval tasks [3], that including query language and expansion. Terrier desktop search can index the different types of file format such as Office documents, PDF files and HTML.

Installing sensei is faster than Terrier and it can work inside sensei or Hadoop environment. Sensei doesn't take a long time to install and it is easy to configure with Hadoop for multi-node taking the advantages of Hadoop features for MapReducing and Hadoop distributed file system.

Fig. 4 notes that Terrier consumes less time to finish the indexing with the several number of machines compared with sensei. And that because of its nature by using posting lists and flush the intermediate data from memory and use the available memory, at the other side Sensei consumed more time for indexing with any number of machines comparing with Terrier and it is using the key and value for indexing data with the assistance of the Hadoop's MapReduce. The observation here the Sensei has sharpness reduce in time through increasing numbers of machines, this maybe lead to crossover point between Sensei and Terrier that can help to find the most efficient indexing strategies.

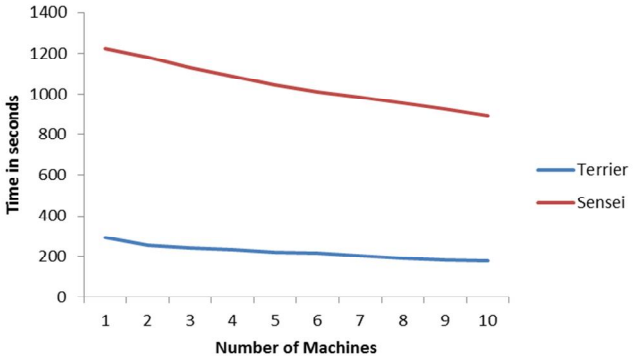

Fig. 4: shows the indexing time for Terrier and Sensei.

The percentage difference used to find the percent different of indexing time between Terrier and Sensei by calculating the average of their indexing consuming time, the percentage difference by them can calculated by the following formula:

$$Percent\ formula = (| (V_1 - V_2) | / ((V_1 + V_2)/2)) * 100 \quad (1)$$

Where V_1 is average indexing consuming time of Terrier, and V_2 is the average indexing consuming time of Sensei by applying the Eq. (1), Terrier's indexing time is better than Sensei and the percentage difference between them is 130 %.

Applying the two strategies on the suggested dataset to find the output that support our hypothesis and measure the output in term of MB/S, the size of used dataset is 113.5MB and it contains 360000 records that can be used in both strategies. To find the best results in both cases, the number of maps is adjust to 12 and the number of reducers to 4, Table 1.

No. of machine	Output of Terrier MB/S	Output of Sensei MB/S
1	0.388115	0.092845
2	0.450454	0.096239
3	0.475885	0.100407
4	0.494114	0.10433
5	0.514574	0.108544
6	0.533605	0.112368
7	0.567072	0.115388
8	0.596687	0.118642
9	0.623089	0.122746
10	0.64662	0.12751

Table 1: Output in MB/S

ISSN 2320-2602

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

Fig. 5 shows the output of the two strategies, it shows the great differentiate between their outputs that measured by MB/S through various numbers of machines.

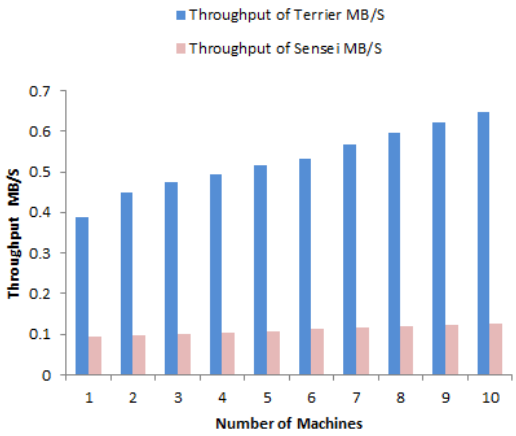

Fig. 5: Throughput of Terrier and Sensei

Measuring speed-up of the two strategies can be calculated by the following formula:

$Sm = T1/Tm$ (2)

Where m is the number of machines and $T1$ is the time consumed by the first machine to create index structure of dataset, and Tm is the execution time in parallel, using m machines [1].

According to the Fig. 6 below the speed-up of Terrier increase by increasing number of machines, same thing happen with the speed-up of sensei, but this show that Terrier has the better speed-up between them.

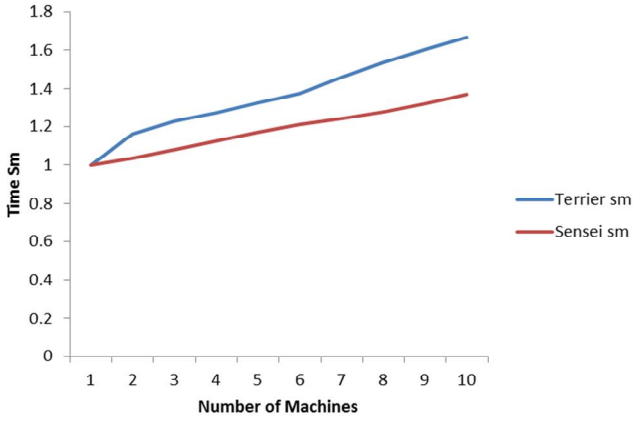

Fig. 6: The speed-up of Terrier and Sensei

According to the results of experiment Terrier is quite faster in indexing data because of using posting list techniques<term, posting list>, which allow it to take the advantages of flushed partial list results and merge them to create final posting list. In sensei using <key,value> pair cannot add more features to its job and also sensei need addition schema of data for treating with different types of data.

CONCLUSION

In this paper, the Information Retrieval (IR) indexing process using the distributed context of MapReduce paradigm was investigated. In particular, both indexing strategies, Sensei and Terrier were discussed. Both supported by Hadoop's MapReduce pattern, that can be applied them on the suggested dataset. First of all, it shows how both strategies deal with large dataset and the efficient of the strategy was studied and analysis according to speed-up. From the experiments Sensei generate much intermediate data which causing slowness of index process. In contrast, per-posting list (Terrier) proved to be the most efficient strategy with large scale dataset. This is because of using the local machine memory and compressing technics of the map-reduce traffic. It is also because Terrier can flush the partial indices to disk when memory is exhausted, and the final index build by merging the flushed indices. The different percentage between the two strategies can be reached to 130%. According to our experiment Terrier is more efficient indexing strategy than Sensei.

REFERENCES

[1] McCreadie, R., Macdonald, C., & Ounis, I. (2012). MapReduce indexing strategies: Studying scalability and efficiency. *Information Processing & Management, 48(5)*, 873-888.

[2] Macdonald, C., McCreadie, R., Santos, R. L., & Ounis, I. (2012). From puppy to maturity: Experiences in developing terrier. *Open Source Information Retrieval*, 60.

[3] Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., & Lioma, C. (2006, August). Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the OSIR Workshop (pp. 18-25).*

[4] He, C., Weitzel, D., Swanson, D., & Lu, Y. (2012, November). Hog: Distributed hadoop MapReduce on the grid. In High Performance Computing, *Networking, Storage and Analysis (SCC), 2012 SC Companion:* (pp. 1276-1283). IEEE.

[5] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium* on (pp. 1-10). IEEE.

International Journal of Advances in Computer Science and Technology (IJACST), Vol.5 , No.3, Pages : 01-06 (2016)
Special Issue of ICDCB 2016 – Held during April 14-15, 2016 in Kuala Lumpur, Malaysia
http://warse.org/IJACST/static/pdf/Issue/icdcb2016sp01.pdf

[6] Heinz, S., & Zobel, J. (2003). Efficient single-pass index construction for text databases. *Journal of the American Society for Information Science and Technology, 54(8*), 713-729.

[7] Yu, W., Wang, Y., & Que, X. (2014). Design and evaluation of network-levitated merge for hadoop acceleration. *Parallel and Distributed Systems, IEEE Transactions on, 25(3),* 602-611.

[8] Abdulkarem, M., & Latip, R. (October, 2015). Data Transmission Performance Analysis in Cloud and Grid. *ARPN Journal of Engineering and Applied Sciences.* VOL. 10, NO. 18.

[9] McCreadie, R., Macdonald, C., & Ounis, I. (2009). Comparing distributed indexing: To MapReduce or not?. *Proc. LSDS-IR,* 41-48.

[10] http://Terrier.org.

[11] http://senseidb.github.io/sensei/overview.html