Computer Science and Information Systems 11(3):1037–1054 DOI: 10.2298/CSIS130920063L

The Efficient Implementation of Distributed Indexing with Hadoop for Digital Investigations on Big Data

Taerim Lee[1], Hyejoo Lee[2], Kyung-Hyune Rhee[1], and Sang Uk Shin[1]

[1] Pukyong National University,
Busan, Republic of Korea
{taeri, khrhee, shinsu}@pknu.ac.kr
[2] Kongju National University,
Gongju, Republic of Korea
hyejoo2010@gmail.com

Abstract. Big Data brings new challenges to the field of e-Discovery or digital forensics and these challenges are mostly connected to the various methods for data processing. Considering that the most important factors are time and cost in determining success or failure of digital investigation, the development of a valid indexing method for efficient search should come first to more quickly and accurately find relevant evidence from Big Data. This paper, therefore, introduces a Distributed Text Processing System based on Hadoop called DTPS and explains about the distinctions between DTPS and other related researches to emphasize the necessity of it. In addition, this paper describes various experimental results in order to find the best implementation strategy in using Hadoop MapReduce for the distributed indexing and to analyze the worth for practical use of DTPS by comparative evaluation of its performance with similar tools. To be short, the ultimate purpose of this research is the development of useful search engine specially aimed at Big Data indexing as a major part for the future e-Discovery cloud service.

Keywords: Electronic Discovery, e-Discovery, Digital Forensics, Evidence Search, Indexing Performance, Hadoop MapReduce, Distributed Indexing.

1. Introduction

Recently, the number of lawsuits is rapidly increasing among business corporations due to the conflict of their interest. These types of incidents can be found easily around us, especially international disputes occur with frequency in the field of patent. But no matter who wins or loses the case, the most important thing is all companies involved in litigation are damaged economically with billions of dollars and they don't even know when the fight is going to be over. The only way to stop this remorseless dispute is that one litigant party secure crucial evidence closely related to the litigation issues and applies it to prove their legitimacy in trial. So, many business owners, professional executives and legal experts start to realize how serious the situation is and they are growing more interested in the skills of e-Discovery or digital forensics.

Electronic discovery (or e-discovery, eDiscovery) refers to discovery in civil litigation which deals with the exchange of information in electronic format called ESI (Electronically Stored Information). This legal system was the subject of amendments to the Federal Rules of Civil Procedure (FRCP), effective December 1, 2006, as amended to December 1, 2010 [14]. In addition, the growing number of legal cases for civil or criminal trials where critical evidence are stored in digital storages has been submitted as the digital forms of information with a high preference. So, state law now frequently addresses issues relating to electronic discovery. These changes make every litigant have a responsibility to produce their own evidence for themselves, so the use of digital forensic or e-Discovery tools becomes a necessary. Furthermore, the appearance of various IT compliances [15] pushes many global companies to reconstruct their business process and adopt a professional e-Discovery service or solution because traditional ERP (Enterprise Resource Planning) solutions are not satisfied enough to cope with fast-growing requirements of IT compliance effectively. To solve this problem, vendors who are related to the field of e-Discovery and forensics have competitively developed and released their own service systems applying the state-of-the-art technologies, but many drawbacks and challenges are still remain. Among them, the most serious problem is about the cost for doing an e-Discovery work.

In general, e-Discovery work is performed by jurists and IT experts who are collaborating with each other. When the litigation is filed, attorneys or a legal team hired by the litigant analyze the contents of petition and figure out major issues at first. And then, they produce a keyword list about evidence which must be secured on the basis of analyzed issues and deliver it to the IT experts. By using this keyword list, IT experts or a team composed of specialists for information retrieval or people of experience at forensic tools search the relevant data as potential evidence and visualize them for review. After this procedure, attorneys can review or analyze again the extracted data from various points of view such as suitability, sensitivity and confidentiality [11] in legal hold[1] situation [16]. According to this explanation, although doing an e-Discovery work sounds so easy, actually this is very complicated work and there are many cases which this procedure is not going well because of several unexpected variables such as system errors, data loss, or technical failures. In particular, electronic information is considered different from paper information because of its intangible form, volume, transience and persistence. Also, electronic information is usually accompanied by metadata that is not found in paper documents and that can play an important part as evidence (for example, the date and time a document was written could be useful in a copyright case). The preservation of metadata from electronic documents creates special challenges to prevent spoliation. For these reasons, developing an almighty system that can solve all kinds of problems is realistically impossible, so vendors focus on the development of functions for helping people deal with various e-Discovery tasks.

Fig. 1 shows the Electronic Discovery Reference Model usually called EDRM [7] and it was designed as part of an effort to provide essential requirements and guidelines of e-Discovery work to the people concerned, especially it also provide reference technologies to vendors. But, this model is too comprehensive and fundamental, so it is not suitable for reflecting the requirements caused by the latest practical problems.

[1] The legal hold is a process to preserve all forms of relevant information from malicious or inadvertent falsification.

Particularly in order to propose a solution for a specific problem, identification of key tasks about this matter and analysis of experimental result from various and concrete cases should be accompanied. As we know, according to some facts about the cost problem, the biggest cost-consuming part is the procedure for the attorney's review and it is bound up with time - the time spent considering whether each document is responsive or not [5]. For sticking to e-Discovery's basic principle that each document must be reviewed by a law expert before it is chosen as evidence and cutting down review expenses at the same time, the effectiveness of information retrieval to decide documents as review target should be improved. In the end, development of effective search method for finding relevant evidence more quickly and accurately is the key to the solution of cost problem. But in recent days, the biggest difficulty for doing this is the problem of Big Data.

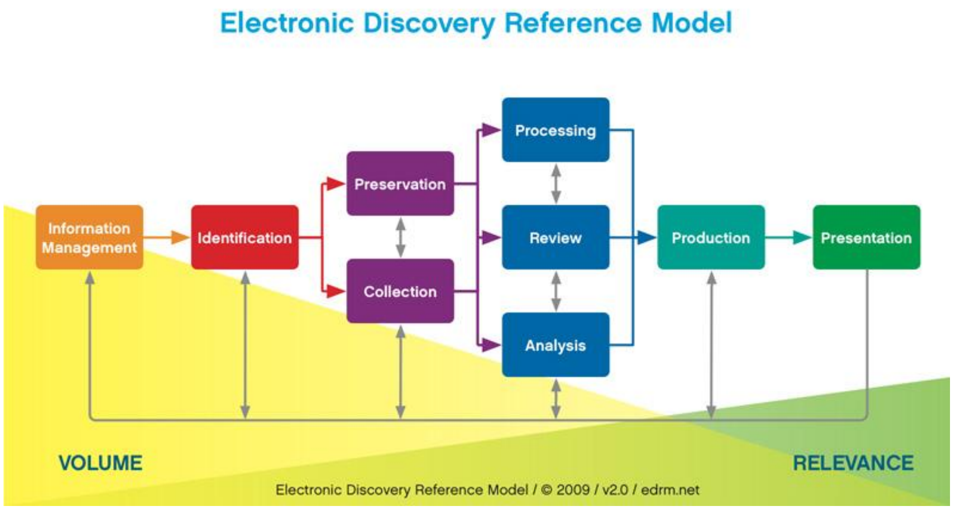


Fig. 1. Electronic Discovery Reference Model [7]

Big Data [17] is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. That means the special techniques must be needed to process the data within a tolerable elapsed time, not excepting the digital investigation case. In other words, Big Data brings new challenges to the field of e-Discovery or forensics as like it does in other research areas and those are mostly connected to data processing methods for capture, curation, storage, search, sharing, transfer, analysis, and visualization. Suppose the litigant has a Big Data as an investigation target and he did not consistently manage them by using a special tool for search, the situation may be getting worse. At a time like this, all tasks for evidence search should be performed extemporarily from start to finish and more unfortunate thing is that such case occurs quite frequently. General text retrieval tools used in the digital forensics at present perform retrieval at an average speed of approximately 20 MB/s with respect to one query. It means 14 hours or more are required to retrieve a query in data of 1 TB [10] and time or cost cannot be expected exactly in case of Big Data. Especially, the most time-consuming job for search is indexing because word vectors of every document have to be created. Of course, there are many ways of search without index, but these methods usually have well-known limitations such as metadata analysis or processing speed. To solve these kinds of problems caused by Big Data, new forms of document processing method for search needs to be established. In 2004, Google published a paper on a process called MapReduce [6]. Its framework provides a parallel programming model and associated implementation to process huge amount of data. This framework is very suitable for handling this kind of job.

This paper, therefore, introduces a Distributed Text Processing System based on Hadoop called DTPS which was belonged in our previous work [12] and explains about the distinctions between DTPS and other similar researches to emphasize the necessity of it. The role of DTPS is basically to make a searchable index files including metadata and it aims to handle this kind of job a lot faster than any other thing. Accordingly, this paper describes a series of experimental results by using different prototypes of DTPS. Each experiment was designed to evaluate the performance of specific prototype version which depends on its design for implementation. Final goal of these experiments is to find the best architecture and implementation strategy of DTPS using Hadoop as a major part for evidence search in the future e-Discovery cloud service. At last, the conclusion of this paper and our future work will be described.

2. Related Works

2.1. Distributed Lucene

Distributed Lucene [4] was the HP's project to create distributed free text index with Hadoop and Lucene in 2008. Given the origins of Hadoop, it is very natural it should be used as the basis of web search engines, a representative case of Big Data processing. It is currently used in Apache Nutch [8], an open source web crawler that creates the data set for a search engine. Nutch is often used with Apache Lucene, which provides a free text index. Despite the link between Hadoop and Lucene [8], at the time of developing there is no easy, off the shelf way to use Hadoop to implement a parallel search engine with a similar architecture to the Google search engine. So, after Doug Cutting came up with an initial design for creating a distributed index using Hadoop and Lucene, HP Labs published the technical report to describe their work for implementing a Distributed Lucene based on this design. For this reason, this project was necessarily undertaken in order to better understand the architectural style used in Hadoop. It means Distributed Lucene had a few limitations in respect of the function because developers only focused on the smooth combination of two other external open source projects, to the exclusion of its performance.

One notable point is Distributed Lucene does not use HDFS directly although the code for it is heavily influence by HDFS and was written by examining the HDFS code. The important reason for doing so it is not possible for multiple clients (or slaves, working nodes) to write the same index in HDFS. Therefore, each client in Distributed

Lucene must create own index about their quotas at first, and then they have to wait their turn for additional updating the same index. But in order to parallelize index creation, it is desirable for multiple clients to be able to access the same index and it is quite feasible through the reconstitution of operation methods for creating index. The other point is about Lucene. Lucene indexes generally contain a number of different files, some of which may be smaller than the 64MB block size for HDFS, so storing them in HDFS may not be efficient. Also, a few limited search techniques provided by Lucene at that time could be used because this project was suspended at the development quality of alpha.

2.2. Katta

Katta [1] is a scalable, failure tolerant, distributed, data storage for real time access. Katta serves large, replicated, indices as shards to serve high loads and very large data sets. These indices can be of different type. Currently implementations are available for Lucene and Hadoop map files. In other words, Katta is recent project adopted latest version of Lucene which provides various functions for using advanced search techniques like machine learning. But, Katta's structure and workflow may does not fit for meeting the growing demands of Big Data in digital investigation.

Fig. 2 shows how Katta works. As you can see, Katta uses an independent cluster or single server for creating index and it uses HDFS or shared file system only for storing the result of indexing. Strictly speaking, Katta is an application for distributed Lucene running on many commodity hardware servers very similar to Hadoop MapReduce, Hadoop DFS, HBase, Bigtable or Hypertable and this means a series of tasks for creating index are not processed dispersively. As a result, the efficiency of indexing and searching is totally influenced by the performance of each slave or cluster which deals with this kind of job. Well, this could be a good way of handling the simultaneous requests from multiple users, but it is not helpful when large amounts of data should be processed at a time. Therefore, Katta suggests a special recommendation and a sample code called SequenceFileCreator to manage this situation. While different and large data sources will probably exist, if user want to leverage the power of Hadoop while indexing SequenceFileCreator could be a good idea to get the data to the Hadoop DFS as raw text or as a sequence file [1]. However, it is just a sample code that will not make the available index, and this means enabling a fully distributed indexing based on Katta is the responsibility of developers using it.

2.3. Forensic Indexed Search System: HFSS

HFSS [10] is the forensic indexed search system which has been developed at Electronics and Telecommunications Research Institute (ETRI) of Korea. It provides a distributed forensic indexing and a web-based Forensic search service using Hadoop because it gives scalability as well as performance improvement. The most remarkable feature of HFSS is how to use the HBase to overcome the limitations about HDFS block size and MapReduce task. In MapReduce, book-keeping information is saved to keep track of the task's status and process during a job initialization. Since a map task takes a block at a time when default input format is used, handling a lot of small files which needs a lot of map tasks causes book-keeping overhead. For this reason, HFSS uses a NAS to store original documents as target of indexing and loads extracted data into HBase table. The way of text processing DTPS try to seek is exactly same with HFSS, but using the HBase to store index information may cause the overhead time when the query is requested for search compared to the general way of using index files directly. Also, users who want to apply the advanced search techniques or who familiar with the traditional ways of search may feel HFSS is too difficult for them because available search method must be kept within bounds of HBase in NoSQL.

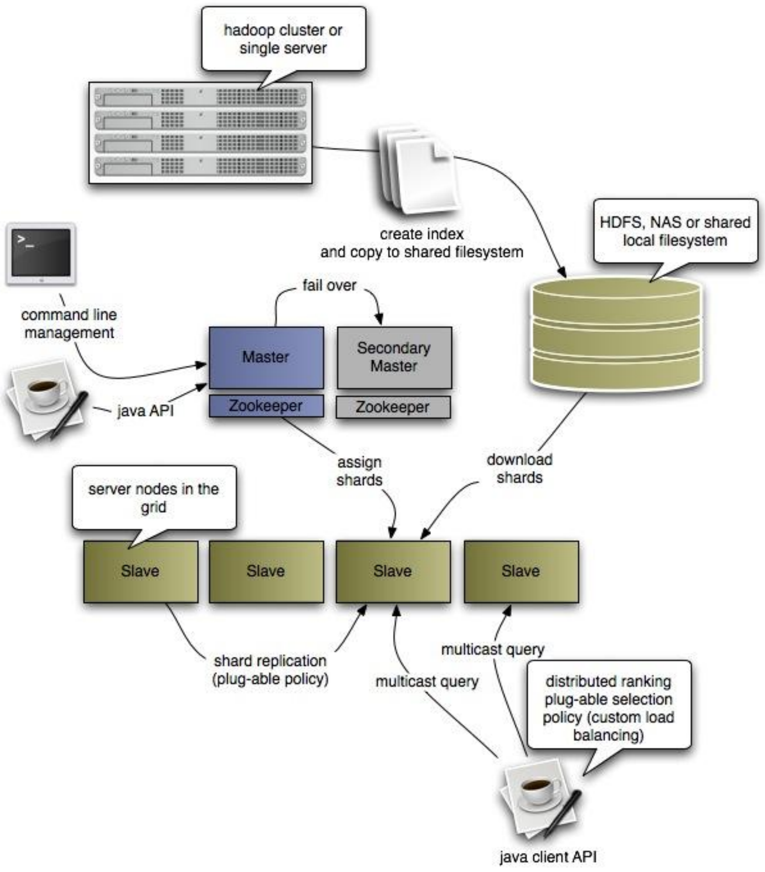

hadoop cluster or single server
create index and copy to shared filesystem
HDFS, NAS or shared local filesystem
command line management
fail over
Master
Secondary Master
Zookeeper
Zookeeper
java API
assign shards
download shards
server nodes in the grid
Slave
Slave
Slave
Slave
shard replication (plug-able policy)
multicast query
multicast query
distributed ranking plug-able selection policy (custom load balancing)
java client API

Fig. 2. About Katta: How Katta works [1]

3. Implementation Method for Efficient DTPS with Hadoop

In this section, the implementation scopes of DTPS prototype will be described. This includes system architecture for overcoming problems mentioned in section 2, basic requirements for text processing, constraints of each function and implementation strategies for the differentiation of performance test.

3.1. Prototype Design of DTPS

Fig. 3 shows the initial design of DTPS. This architecture was designed by considering the actual service type for digital investigation and combination with e-Discovery cloud service in the future. According to the service process, a simple use scenario and workflow of DTPS are as follows. Naturally, additional functions for doing this were added.

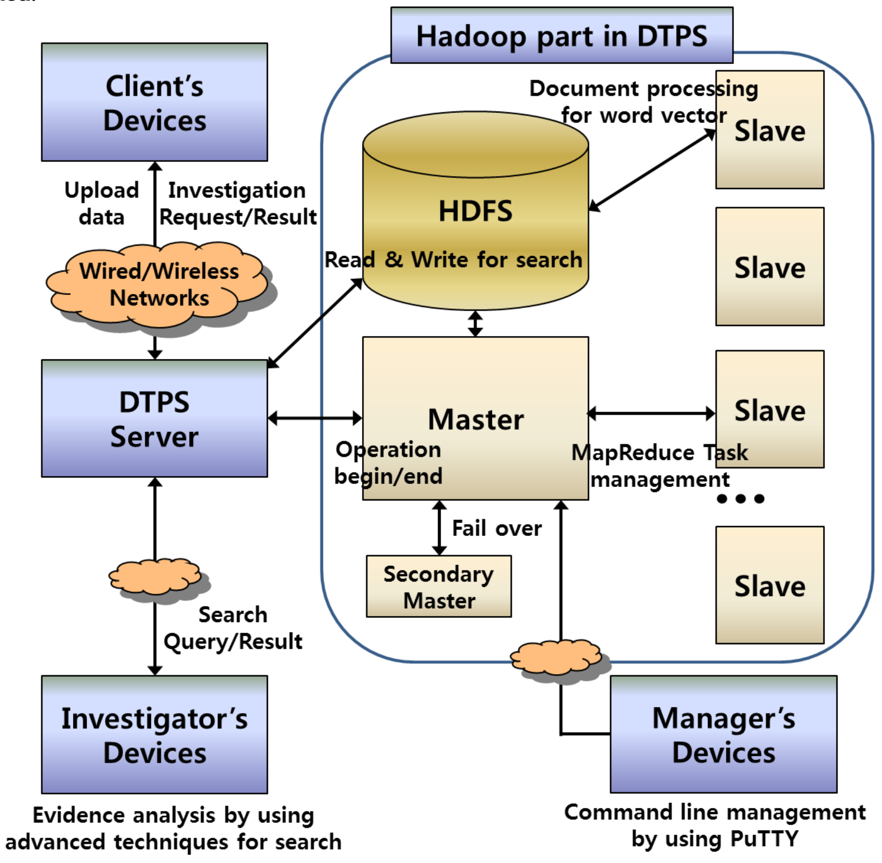


Fig. 3. Initial Design and Workflow of DTPS

When the clients call for an investigation at the beginning, they start to upload a target data of investigation to the server's local file system. After successful completion of uploading, making input files in HDFS and extracting metadata runs in parallel, and DTPS server gives the order to the master for starting the operation of text processing. At the word of command, the master and slaves carry out their own tasks as programmed by MapReduce for creating word vector representations, and work results are stored in HDFS again. DTPS manager can monitor this whole process and perform various tasks for the management of Hadoop configurations through the command line tool like PuTTY or DTPS server application. Meanwhile in order to find evidence, investigators create a series of queries with advanced search techniques and send them to the DTPS server. And then, server returns the results of search to the investigator for analyzing and selecting some crucial evidence. Finally, DTPS server returns the result of investigation to the clients for the future trial.

3.2. Basic Requirements

The role of Hadoop MapReduce in DTPS prototype is restricted only for making a vector representation of document, and it is the most important and time-consuming task for creating index. These indices can be used for various search operations from the simple Boolean method to the advanced methods (for example, ranked search and document classification using machine learning algorithms). Also, DTPS uses HDFS directly to save its input documents and outputs. Outputs consist of six files, and descriptions about these files are shown in Table 1.

In Table 1, there are two files which are optionally created for supporting the advanced search techniques of DTPS. The first is XML file for archiving the metadata information of documents. In some cases, the metadata may be more important than the contents of document as evidence, and patent infringement case could be a good example like this. Therefore, considering that metadata search is the one of the helpful functions for e-Discovery solution, the implementation of it on DTPS takes priority above everything else. The metadata information extracted by DTPS is as follows and this is the simple design for gathering common elements from various document formats.

```
<DTPS_Metadata>
  <document>
   <docid>E:\ComSIS_EDaaS_cam_rdy.doc</docid>
   <metadata>
    <Author>Taerim Lee</Author>
    <creator>Taerim Lee</creator>
    <Creation-Date>2013-04-03T00:52:00Z</Creation-Date>
    <Last-Save-Date>2013-04-03T01:56:00Z</Last-Save-Date>
    <Last-Modified>2013-04-03T01:56:00Z</Last-Modified>
   </metadata>
  </document>
  <document> ... more documents </document>
</DTPS_Metadata>
```

Table 1. Summary of DTPS Index File Extensions

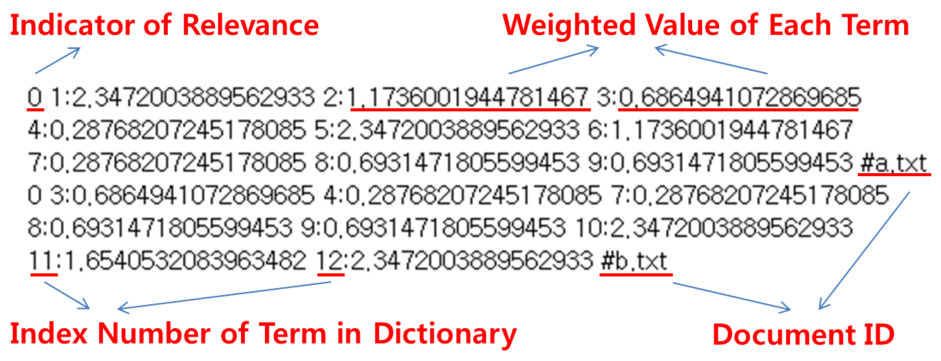
Name	Extension	Brief Description
Term Dictionary	.tdf	The term dictionary, stores term list (Index number of each term, all terms in document collection).
Document List	.dlf	The document list, stores document information (Index number of each document, all docIDs in collection, docID is the original path of each file before it was collected from its local file system).
Posting List	.plf	The posting list, stores term information(Same index number of term in .tdf, Total Term Frequency in collection, Document Frequency, the list of value pairs {document index number : Local Term Frequency}, Local Term Frequency is the frequency number of specific term in one document).
Document Vector	.dvf	The vector representations of all documents in collection (Same index number of document in .dlf, the list of value pairs {term index number : weighted term vector}, weighted term vector is calculated by TF-iDF method).
Metadata	.xml	The common metadata information of all documents in collection (docID, author, creator, creation-date, last-save-date, last-modified).
Training Set	.dat	The training examples for document classification using machine learning method in specific category. This file is created by modifying the .dvf file, additionally stores indicator values of relevance.

This is the example of extracted metadata from MS Word file and the XML tags apply to the original names of metadata as it is. The reason why we use the XML format is to consider the flexible extension of metadata structure caused from the additional information requirements. Apart from the indexing process, metadata extraction will be performed in the beginning process of DTPS in order to upload indexing target documents to the HDFS. So, this file is the first outcome produced by DTPS. The second is DAT file for training set and this file will be created by modifying the .DVF (Document Vector File). DTPS uses SVM^{light} [9] for document classification as an advanced search method based on machine learning. Therefore, DAT file is exactly same with input format of SVM^{light} so modifying the .DVF means adding target values of relevance(+1 as the target value marks a positive example, -1 a negative example respectively) to the document vector representation. Fig. 4 shows the example of training set partly extracted from the second file.

Above this, essential functions of DTPS programmed in Hadoop MapReduce for creating index are as follows.

– Document Loader: The step for loading individual documents from storage given by the source name of document information. Loading refers to the operation of opening a stream to fit for specific file format of each document. But, this prototype takes care of two types of file stream for local file system and HDFS.

– Feature Extraction [13]: The step for converting the document into clear word format called term dictionary. So, tokenization and removing stop words is performed.
– Feature Selection [13]: The step for selecting a subset of relevant features for use in model construction. This model is a vector space written by statistical characteristics of language. For prototype, best known scoring scheme of TF-iDF was applied with the log frequency weight of term and cosine normalization.


Indicator of Relevance
Weighted Value of Each Term
Index Number of Term in Dictionary
Document ID

```
0 1:2.3472003889562933 2:1.1736001944781467 3:0.6864941072869685
4:0.28768207245178085 5:2.3472003889562933 6:1.1736001944781467
7:0.28768207245178085 8:0.6931471805599453 9:0.6931471805599453 #a.txt
0 3:0.6864941072869685 4:0.28768207245178085 7:0.28768207245178085
8:0.6931471805599453 9:0.6931471805599453 10:2.3472003889562933
11:1.6540532083963482 12:2.3472003889562933 #b.txt
```

Fig. 4. The Example of DTPS Output

3.3. Implementation Strategy for Differentiated Experimental Design

Basic tasks in DTPS with MapReduce are as follows.

1. Preparation: Get input files ready in HDFS and extract metadata from the original documents (.xml)
2. Job Configuration: Set input paths of HDFS to decide the processing unit and the number of MapReduce tasks
3. Map: Reading the contents of each document - Tokenization - Stopword Filtering - Creating {Key, Value} with {docID, Term}. The docID is a path and name of the document in local file system, and term is a tokenized word in a currently processed document. The docID can be basically extracted by using the configure function in org.apache.hadoop.mapred.MapReduceBase.
4. Reduce: Comparing {docID, Term} - Counting a term frequency in same document and a document frequency in same collection - Writing the term dictionary (.tdf), the document list (.dlf), and the posting list file (.plf)
5. Completion: Calculating a TF-iDF weight - Creating a vector representation (.dvf) file of all documents in collection, and SVM input file (.dat) with the additional option

In order to confirm what the best implementation method for DTPS is, five test cases were established and differentiated factors in implementation for each case were applied. Details about experiments are summarized in Table 2.

Table 2. The summary of each experiment (For the description of differentiated factors, each number for basic tasks was simply used on the behalf of its name. - 1. Preparation 2. Job Configuration 3. Map 4. Reduce 5. Completion)

No	Purpose	Differentiated factors of each case in implementation
Case 1	Basic Test (Comparison Group)	1. Just copy input files from server (local file system) to HDFS. 2. Processing Unit: Each file in document collection. The number of map tasks is same with the number of files. 3. Reading one line in each document and processing it. 4 & 5. No different from the basic tasks
Case 2	Memory Test	1, 3, 4 & 5. Same with Case 1 2. In order to avoid a possible failure caused by the book-keeping overhead in Case 1, add some codes to define a new text input format for processing multiple files like the MultiFileWordCount example in Hadoop. The number of map tasks is two. Additional factors for execution are : - Running DTPS in Hadoop with increasing the heap size of Java. - Running DTPS in Hadoop with adding physical RAM to the master.
Case 3	Compression Test	1 & 5. Same with Case 1 2. Same with Case 2 fundamentally, but add some codes to the Hadoop Job configuration for setting a compression library, Snappy. The number of task is same with Case 2. 3. Reading one line in each document and creating {docID, Term} with tokenization. After that, compress the outputs of mapper by using a Snappy before the reduce step. 4. Decompressing the mapper's outputs, and then start reduce task.
Case 4	Merge Test	1. To make one file for MapReduce input in HDFS, the content of each document was extracted, and merged to the input.xml. It is the same way as the metadata extraction. 2. Processing unit for inputs: created input file. The number of map tasks is the quotient (total size of created input file divided by Hadoop block size) + 1, same with the total number of used blocks for saving this input file in HDFS. 3. There are two primary XML tags, docID and content. After getting the docID at first, and then use the content to make pairs with tokenized terms. Actually, each value of the content tag is the full text of one document. 4 & 5. Same with Case 1
Case 5	Compression with Merge Test	This is the combination case between 3 and 4. 1. Same with Case 4 2. Same with Case 3, but the number of task is only one like Case 4. 3, 4 & 5. Same with Case 3

Case 1 is a basic test with no additional conditions as a comparison group. So, it will be performed by using a default configuration of Hadoop and document collection as it is. On the other hand, Case 2 is for comparing the performance according to the Hadoop configurations about memory and it has special meaning to suggest an alternative when experiment for Case 1 is failed because of memory overflow in Big Data processing. To do this, codes for processing multiple files were additionally implemented, and it makes only two map tasks will be required. Well, Case 3 was established to know the different performance of Hadoop when a specific library for compression was applied. Various libraries can be used for this test such as LZO, gzip or b2zip, but Snappy was only considered in our experiment because of its well-known advantages [3]. In Case 4, in order to extract the information of accurate docID from the merged input file, extraordinary measures are required because all files in HDFS are stored separately based on its block size and merging let the file lose its own metadata. Therefore, implemented code for merging writes one XML file with special tag for keeping this kind of information every time the new document is added. In our prototype, only two tags are used for input.xml, docID and content. It is because other metadata will be extracted and merged separately through the preparation task.

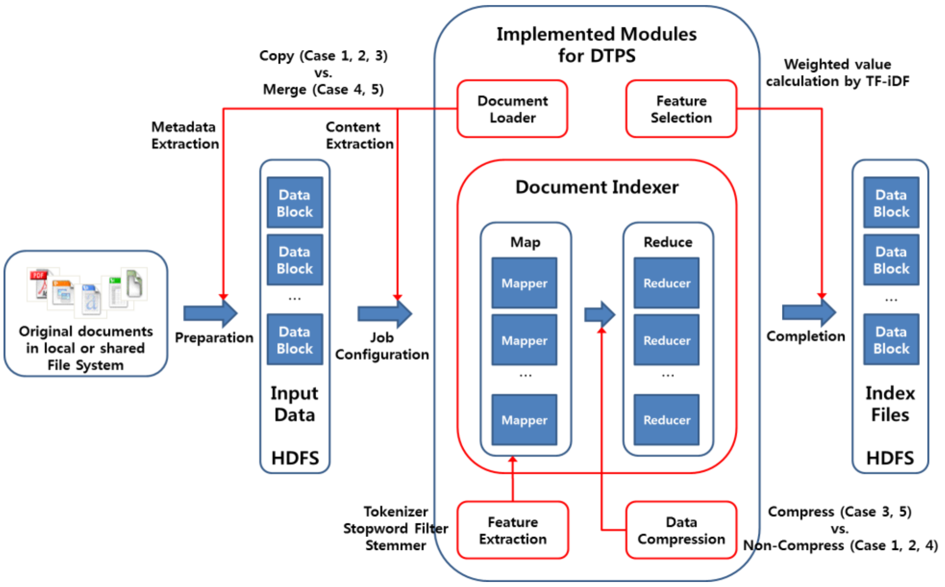


Fig. 5. The implemented modules of DTPS and control flow for each different experiment

Fig. 5 shows the implemented modules of DTPS based on the basic requirements for creating index and control flow for applying the features of each experiment in Table 2. Document Loader and Data Compression can be divided into separate parts, which can perform several different roles in this control flow. For example, Loader has a document parser for data extraction, and it is used for Preparation stage in the case of merge test, but it will be used for Job Configuration and Map stage in the case of copy test. So, they would be the biggest influence to the experimental results.

4. Evaluation Results and Analysis

Document collection used in our experiments is the EDRM Enron Email Data Set v2 [2]. This collection consists of 685,979 .txt files in 159 directories and the total size is about 4GB (3,991,162,863 bytes). Each .txt file was made by data extraction from email and its attachment files. For wide use of it, there are more types being provided by EDRM, such as PST or XML version.

4.1. Configurations of DTPS

Environment for developing DTPS and configurations of test-bed are as follows.

– System Hardware
  1) Hadoop Master: Two different masters for the memory test in Case 2. One of the masters has a 4GB RAM with default size of java heap which is generally used in PC, and another master has a 6GB RAM with 2GB maximum size of heap.
  2) Hadoop Slave: Two slaves with same hardware devices. Each slave has a Core 2 Duo CPU and 4GB size of RAM.
– OS: Ubuntu 12.04 LTS 64bit for the availability of extended RAM size
– IDE: Eclipse Standard, Kelper Release
– Programming Language: Java-6-oracle 1.6.0_45 version
– Library: Apache Hadoop 1.0.4 , Apache Tika 1.4 , Snappy 1.1.0 , SVM^{light}

Apache Tika was used for detecting and extracting metadata and structured text content from various documents using existing parser. Compared to the time of preparation task (Case 4 merging vs. Case 2 non-merging), the performance of Tika could affect our experiments. But, SVM^{light} was applied for practical use of DTPS in order to give an example as an advanced search based on machine learning. It, therefore, has nothing to do with our experiments at all.

4.2. Test Results and Analysis

Table 3 shows the result of each experiment. The performance of DTPS was compared by the time of job completion for text processing and each test was repeatedly performed at least three times to find out the effects of initial job and different conditions of slaves such as communication status, unexpected errors or Hadoop fail over. It is generally recognized that the first job of Hadoop after it has started takes more time for warming up than the second or further executions although it is a same job.

Interestingly, while there was no real effect of Hadoop initial job, master's fail over for unexpected slaves' error would be the biggest influence to the completion time. In Case 1, all experiments were failed because of java errors associated with memory overflow. As we mentioned earlier in section 2 that too many tasks in MapReduce cause the book-keeping overhead, so failing job is a natural outcome in this case and additional actions are required to handle this problem like the rest of experiments. On the contrary, text processing job succeeded without any problems in Case 2, but there is no great difference according to the RAM size. It means the memory size in Hadoop is more important factor for guessing the success or failure of a specific MapReduce job than its performance. Consequentially, considering a much bigger collection than used in our experiment, blindly increasing the size of RAM is a leap in the dark and will not help. In Case 3, there seem to be no advantages of Snappy in processing speed. But, Table 4 shows it was effective enough to improve the MapReduce performance on the other side.

Table 3. The time of job completion in each experiment

Number of tries / Test Cases	Case 1	Case 2	Case 3	Case 4	Case 5
The first try	Failed	4h,38m,41s	4h,30m,19s	8m,54s	8m,43s
The second try	Failed	4h,25m,25s	4h,26m,55s	8m,41s	8m,29s
The third try	Failed	5h,2m,29s	4h,25m,7s	8m,45s	8m,55s
Average time except the first try	N/A	4h,43m,57s	4h,26m,1s	8m,43s	8m,42s
Average time to prepare input	4 hours			2 hours	
Total time of completion	N/A	8h,43m,57s	8h,26m,1s	2h,8m,43s	2h,8m,42s

Table 4. The Part of Hadoop Log Information : Non-Using vs. Using Snappy

	Counters	Non-use	Use
FileSystem Counters	FILE_BYTES_READ	7,148,098,440	2,579,056,504
	FILE_BYTES_WRITTEN	8,122,898,403	3,086,829,404
Map-Reduce Framework	Map output bytes	31,193,534,663	31,193,534,663
	Reduce shuffle bytes	969,587,271	502,559,596

As you see in Table 4, Snappy optimizes the distribution of Map outputs for decreasing the number of times being read and written by system I/O. It means Snappy enables MapReduce tasks to be processed more smoothly because the probability of system fail over is relatively lessened. We guessed the reason why the job completion time actually makes no difference with Snappy non-use case is the use of gigabit LAN for fast communication between master and slaves in DTPS test-bed. But, in other situations as Mapper makes a tremendous amount of output by using a bigger collection than our experiment or Hadoop has a poor network environment, the compression library would be very necessary. Before everything else, the deadly problem is the wasted space of storage in all preceding cases because general document is much smaller than HDFS block size. Naturally, Case 4 solved this problem and produced the most notable result in all experiments. Considering the possible overhead time caused by merging to make one input file with tagging information, there have been substantial improvements in the processing speed. In fact, the merging takes less time to prepare input file compared to uploading a full document collection to HDFS, and that means there is no overhead. Finally, from the experimental result of Case 5, we can confirm the best strategy of implementation for DTPS is the combination of making an integrated input file and compressing intermediate data processed in MapReduce task.

And now let's take the final evaluation result from the comparison between three different simulations, Lucene indexing on a single machine, Katta indexing with the SequenceFileCreator (same conditions with the hardware configurations of DTPS), and DTPS indexing. Actually, we made a new code for SequenceFileCreator by modifying a sample code provided by Katta. Because this code only made several random records in sequence file format from one text file in order to recommend one way of using Hadoop for distributed indexing. That means it cannot process multiple files as general indexing tools did and does not fit for our test. On the contrary, our modified SequenceFileCreator can convert all texts in document collection to sequence file records, but search result produced by using these records is not available because it does not know which documents are including specific contents that match the user's information request. So, the goal of this trial is just to compare the work time required for preparing input file between Katta and DTPS. Additionally, for the rapid progression of this assessment, the master was replaced to the better system (Quad-Core i7 CPU, 12 GB RAM) based on the time result in our previous experiments and the same document collection was used.

Table 5. The performance comparison on the indexing speed between Lucene, Katta and DTPS

Average Time \ Cases	Single Lucene	Katta	DTPS
Preparing input	N/A	18m	40m
Indexing	1h, 31m	20m	7m
Additional time	N/A	4m	N/A
Total	1h, 31m	42m	47m

In Table 5, only Katta needs additional time for deploying and adding the Lucene analyzer to its index and total indicates the overall time demanded by each system for being ready to search. Single Lucene means the basically same case with the Katta indexing, non-using the SequenceFileCreator and just copy indices from the independent content server to the HDFS for search. So, it takes the longest time in comparison with the others even if the time for copying index will be excluded. Meanwhile, DTPS takes a little more time than Katta, but it does not matter considering the index of Katta is not available in this experiment as above mentioned. As a result, if the Katta’s SequenceFileCreator writes more information with modified data structure like the input of DTPS to make available index, we can expect the time required by both systems will be nearly the same. Also, we can be sure that merging all texts from multiple documents into one input file is a most useful way of improving the performance of distributed indexing in Hadoop. What was interesting about Katta, it uses only a Map task because Lucene provides the way of incremental indexing for merging distributed index files. Although additional verification of which method is better to make a final index, merging outputs by Reduce task in DTPS or incremental indexing by Lucene in Katta, may be needed, but at least DTPS is the winner in this experiment.

5. Conclusions

This paper described the research for implementing a Distributed Text Processing System using Hadoop MapReduce. Considering the latest requirements of e-Discovery caused by Big Data problems, major object of DTPS was the development of indexing method for search in order to find relevant evidence more quickly and accurately from large-scale data. To do this, five experiments were performed manipulating the code of MapReduce, the memory size of java heap and the type of input. As a result, we confirmed that the best strategy of implementation for DTPS is the combination of making an integrated input file and compressing data processed in MapReduce task. Also, in order to compare the performance of DTPS with similar tools, additional experiment was conducted and the result showed DTPS is useful enough to carry out a series of work for indexing effectively. On the guess that three different projects introduced in the section of related works may be undesirable for processing the Big Data according to circumstances in digital investigation, we hope this paper can clearly give the direction on developing the advanced e-Discovery or digital forensic service. From now on, considering the additional requirements of DTPS for using as the e-Discovery cloud service, complete realization and research on the accuracy improvement of search will be our future work.

Acknowledgement. This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No.2011-0029927).

References

1. Katta. 101tec, Inc. (2010), [Online]. Available: http://katta.sourceforge.net/ (current August 2013)
2. Trec legal track: Identification and download helpers for edrm enron v2 dataset. Text REtrieval Conference (2010), [Online]. Available: http://trec-legal.umiacs.umd.edu/corpora/trec/legal10/ (current August 2013)
3. Snappy, a fast compressor/decompressor. Google Project Hosting (2013), [Online]. Available: https://code.google.com/p/snappy/ (current August 2013)
4. Butler, M.H., Rutherford, J.: Distributed lucene: A distributed free text index for hadoop. Hewlett-Packard Development Company, L.P. (2008), [Online]. Available: https://www.hpl.hp.com/techreports/2008/HPL-2008-64.pdf (current August 2013)
5. Cohen, A.I., Kalbaugh, E.G.: ESI Handbook: Sources, Technology and Process. Aspen Publishers, New York City, USA (2008)
6. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Proceedings of the Sixth Symposium on Operating System Design and Implementation. pp. 1–14. Google Research Publications, San Francisco, USA (2004)
7. EDRM: Edrm framework guides. EDRM LLC (2006), [Online]. Available: http://www.edrm.net/resources/guides/edrm-framework-guides (current August 2013)
8. Hatcher, E., Gospodneti, O.: Lucene in Action. Manning Publications Co., Greenwich, Connecticut, USA (2004)
9. Joachims, T.: SVM^{light}, support vector machine. Cornell University Department of Computer Science (2008), [Online]. Available: http://svmlight.joachims.org/ (current August 2013)

10. Lee, J., Un, S.: Digital forensics as a service: A case study of forensic indexed search. In: Proceedings of the ICT Convergence (ICTC), 2012 International Conference on. pp. 499–503. IEEE, Jeju Island, Republic of Korea (2012)
11. Lee, T., Kim, H., Rhee, K.H., Shin, S.U.: Design and implementation of e-discovery as a service based on cloud computing. Computer Science and Information Systems, 10(2), 703–724 (2013)
12. Lee, T., Kim, H., Rhee, K.H., Shin, S.U.: Implementation and performance of distributed text processing system using hadoop for e-discovery cloud service. Journal of Internet Services and Information Security, Vol.4, No.1, pp. 12-24 (2013)
13. Manning, C.D., Raghavan, P., Schtze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, England (2008)
14. Smith, L., et al.: Federal rules of civil procedure. U.S. GOVERNMENT PRINTING OFFICE (2012), [Online]. Available: http://www.uscourts.gov/uscourts/rules/civil-procedure.pdf (current August 2013)
15. Tarantino, A.: Compliance Handbook (Technology, Finance, Environmental, and International Guidance and Best Practices). John Wiley & Sons Inc., New York City, USA (2007)
16. Volonino, L., Redpath, I.J.: e-Discovery For Dummies. John Wiley & Sons Inc., New York City, USA (2009)
17. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, California, USA (2012)

Taerim Lee received his Bachelor and Master of Engineering degrees from Pukyong National University, Busan Korea in 2008 and 2010, respectively. He is currently doing a Ph.D. program in Department of Information Security, Graduate School, Pukyong National University. His research interests include digital forensics, e-Discovery, cloud computing, and machine learning.

Hyejoo Lee received her M.S. and Ph.D degrees from PuKyong National University, Busan, Korea in 1997 and 2000, respectively. She worked as a senior researcher in Electronics and Telecommunications Research Institute, Daejeon, Korea from 2001 to 2005. She is currently working as Post Doctor in Department of Applied Mathematics at Kongju National University, Gongju, Korea. Her research interests include digital rights management, digital watermarking, multimedia protection and image processing.

Kyung-Hyune Rhee received his M.S. and Ph.D. degrees from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea in 1985 and 1992, respectively. He worked as a senior researcher in Electronic and Telecommunications Research Institute (ETRI), Daejeon, Korea from 1985 to 1993. He also worked as a visiting scholar in the University of Adelaide in Australia, the University of Tokyo in Japan, the University of California at Irvine in USA, and Kyushu University in Japan. He has served as a Chairman of Division of Information and Communication Technology, Colombo Plan Staff College for Technician Education in Manila, the Philippines. He is currently a professor in the Department of IT Convergence and Application Engineering, Pukyong National University, Busan, Korea. His research interests center on multimedia security and analysis, key management protocols and mobile ad-hoc and VANET communication security.

Sang Uk Shin (Corresponding author) received his M.S. and Ph.D. degrees from Pukyong National University, Busan, Korea in 1997 and 2000, respectively. He worked as a senior researcher in Electronics and Telecommunications Research Institute, Daejeon Korea from 2000 to 2003. He is currently an associate professor in Department of IT Convergence and Application Engineering, Pukyong National University. His research interests include digital forensics, e-Discovery, cryptographic protocol, mobile and wireless network security and multimedia content security.

Received: September 20, 2013; Accepted: January 17, 2014.