Implementation and Performance of Distributed Text Processing System Using Hadoop for e-Discovery Cloud Service*

Taerim Lee, Hun Kim, Kyung Hyune Rhee, and Sang Uk Shin†
Pukyoung National University, Busan, Republic of Korea
{taeri, mybreathing, khrhee, shinsu}@pknu.ac.kr

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 of data processing. Considering that the most important factors are time and cost in determining success or failure of digital investigation, development of search method comes first to more quickly and accurately find relevant evidence in 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 similar researches to emphasize the necessity of it. In addition, this paper describes experimental results to find the best architecture and implementation strategy for using Hadoop MapReduce as a major part of the future e-Discovery cloud service.

Keywords: Electronic Discovery, e-Discovery, Digital Forensics, Evidence Search, Hadoop Performance, MapReduce Programming, Distributed Text Processing

1 Introduction

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 also addresses issues relating to electronic discovery. These changes force every litigant to 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 compliance[10] push 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 or software 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

*This work was partly supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Plannig (No.2011-0029927) and by the IT R&D program of MOTIE/KEIT[10035157, Development of Digital Forensic Technologies for Real-Time Analysis].

†Corresponding author: Room 1314, Building 1, Department of IT Convergence and Application Engineering, Daeyeon Campus (608-737) 45, Yongso-ro, Nam-Gu. Busan, Republic of Korea, Tel: +82-(0)516296249

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 in legal hold situation[15][1]. Although doing an e-Discovery work sounds so easy according to this explanation, 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.

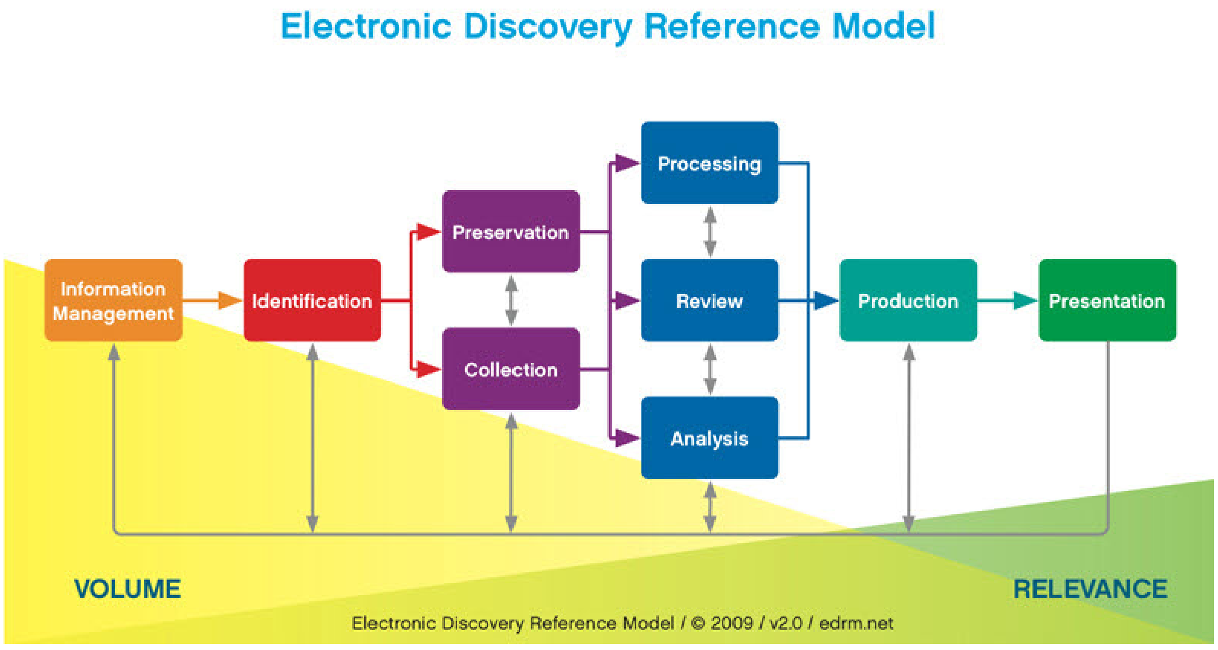


Figure 1: Electronic Discovery Reference Model[9]

Figure 1 shows the Electronic Discovery Reference Model usually called EDRM[9] 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. 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[3]. 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.

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,

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

sharing, transfer, analysis, and visualization[16]. 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[6] and time or cost cannot be expected exactly in case of Big Data.
This paper, therefore, introduces a Distributed Text Processing System based on Hadoop called DTPS which was belonged in our previous work[7] and explains about the distinctions between DTPS and other similar researches to emphasize the necessity of it. Also, this paper describes implementation strategies of DTPS and 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, conclusions of this paper and our future work will be described.

2 Related Work

2.1 Distributed Lucene

Distributed Lucene[2] 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[13], 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[12], 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 that 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 is because 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 additionally 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 and 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.

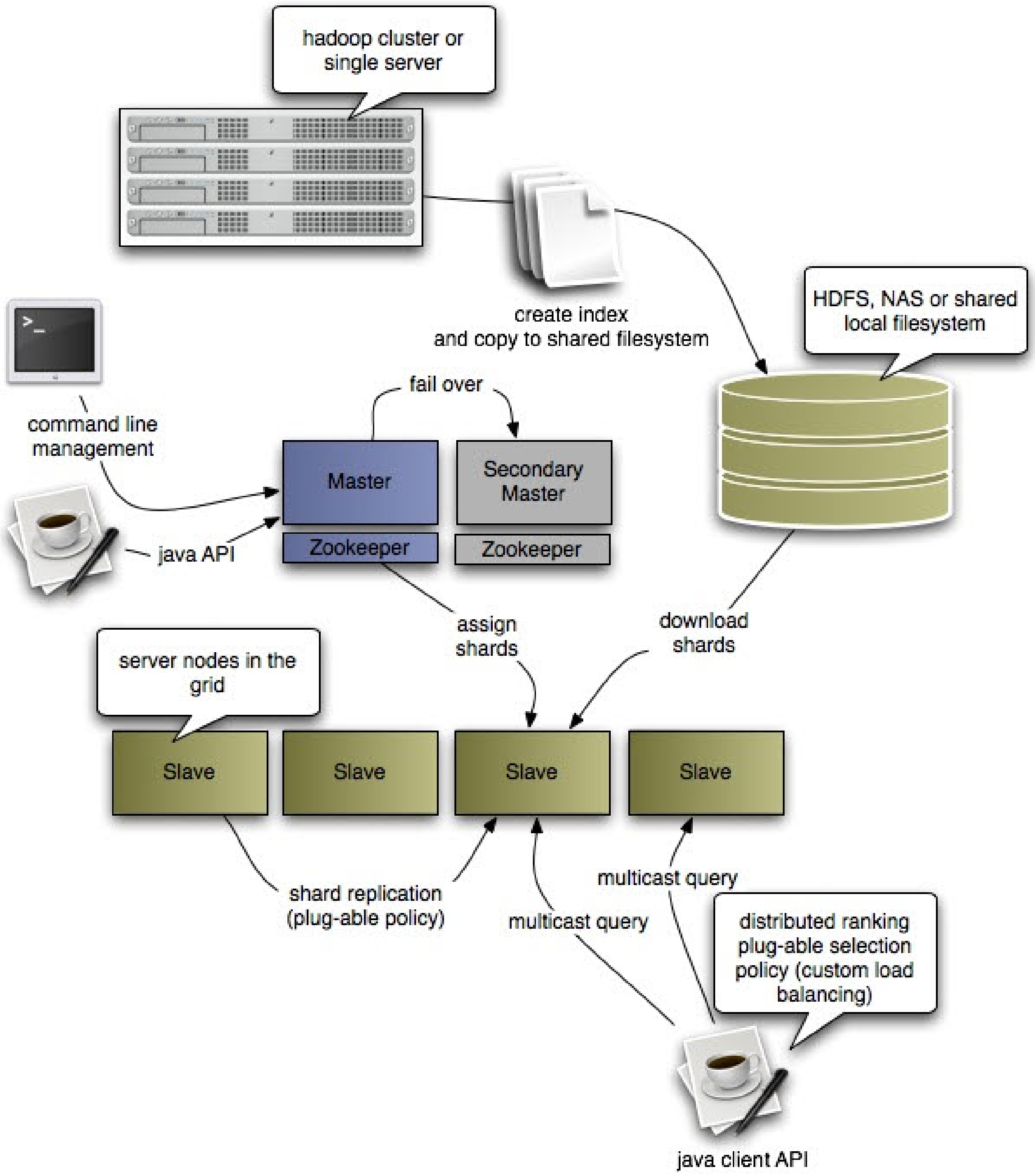

hadoop cluster or single server
create index and copy to shared filesystem
HDFS, NAS or shared local filesystem
command line management
fail over
Master
Secondary Master
Zookeeper
Zookeeper
java API
assign shards
download shards
server nodes in the grid
Slave
Slave
Slave
Slave
shard replication (plug-able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

Figure 2: About Katta : How Katta works[1]

Figure 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 a distributed application running on many commodity hardware servers very similar to Hadoop MapReduce[4], Hadoop DFS, HBase, Bigtable or Hypertable and this means a series of tasks for creating index are not processed dispersively in Katta. 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.

2.3 Forensic Indexed Search System : HFSS

HFSS[6] 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 tasks' 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.

3 Implementation of DTPS based on Hadoop

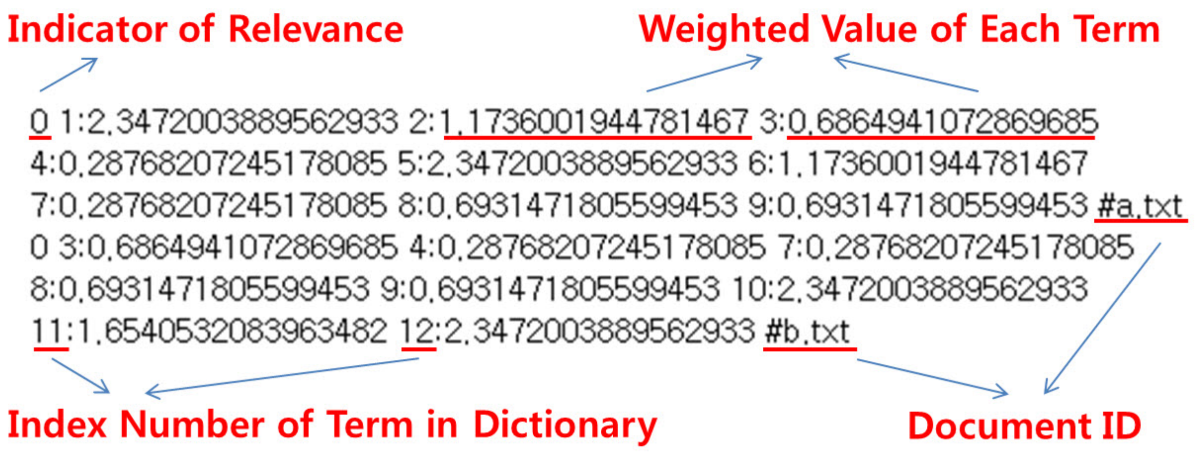
In this section, the implementation scopes of DTPS prototype will be described. This includes basic requirements for text processing, system architecture for overcoming problems mentioned in Related Work, constraints of each function and implementation strategies for the differentiation of performance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

3.1 Basic Requirements

The role of prototype is restricted only for making a vector representation of document, and it is the most important and time-consuming task for creating index. DTPS uses HDFS directly to save its input documents and outputs. Outputs consist of two files, one is the file for term dictionary and another is the file for collection of document information applied term weight. Figure 3 shows the example of DTPS output partly extracted from the second file.
To do this, essential functions of DTPS programmed in Hadoop MapReduce 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 only takes care of the general document format in .txt and uses two types of file stream for local file system and HDFS.

- Feature Extraction[8] : The step for converting the document into clear word format called term dictionary. So, tokenization and removing stop words is performed.


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
```

Figure 3: The Example of DTPS Output

- Feature Selection[8] : 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.

3.2 Initial Design of DTPS

Figure 4 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.
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, DTPS server makes input files in HDFS and 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 by using 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.

3.3 Implementation Strategy for Differentiated Experimental Design

In order to effectively process the Big Data using Hadoop, the most important consideration is not the total quantity of data but the form of input data. Document collection as an investigation target generally consists of numerous small files with various file formats in real e-Discovery case, so the performance of Hadoop could be affected by how to handle this kind of situation. As we know that handling a lot of small files which requires as much as map tasks causes book-keeping overhead, so we should get the best implementation strategy through the performance experiments for properly using a MapReduce in DTPS. Basic tasks for document processing in MapReduce are as follows.

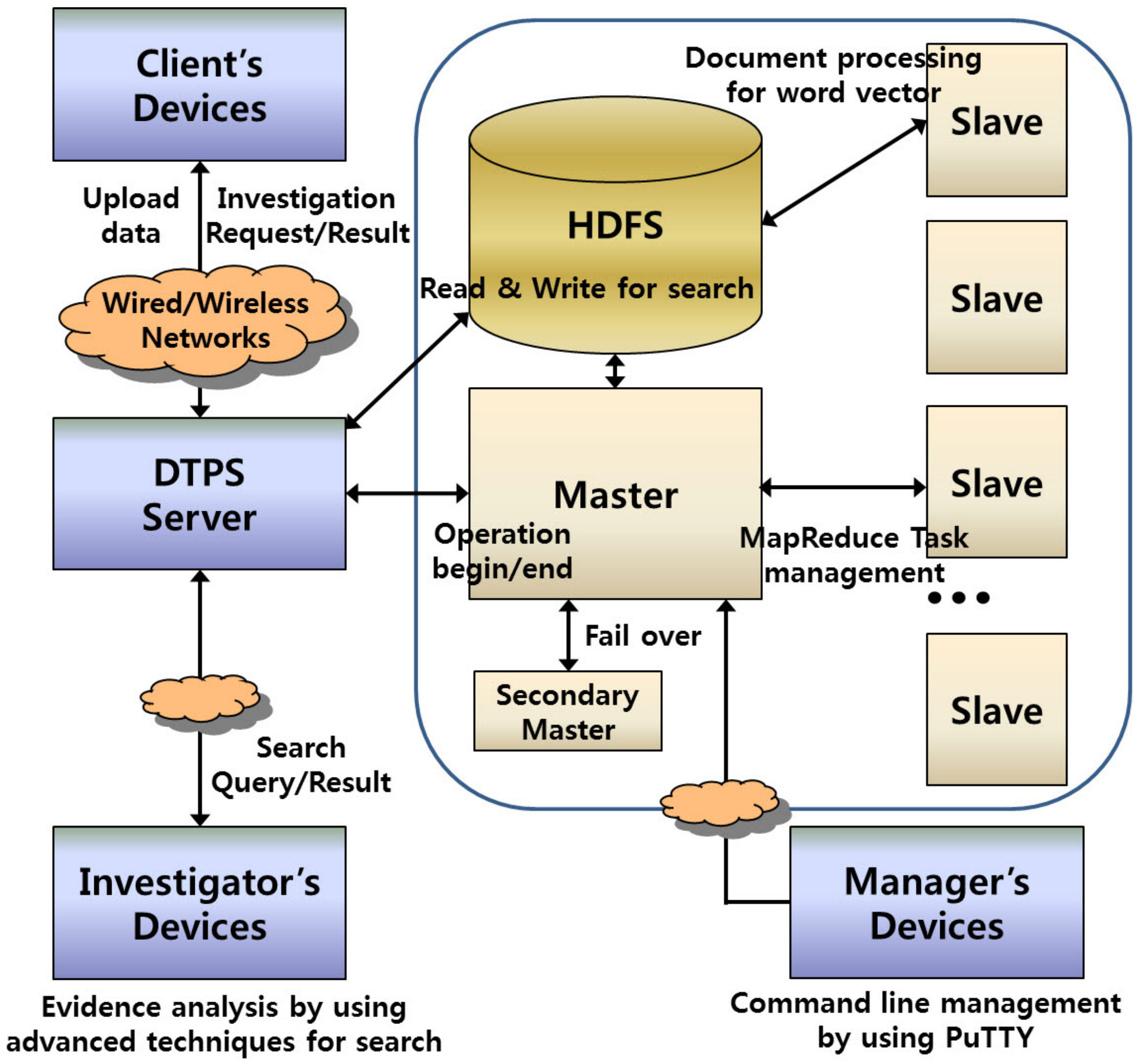


Figure 4: Initial Design and Workflow of DTPS

1. Preparation : Get input files ready in HDFS

2. Job Configuration : Set input paths to decide the processing unit and the number of tasks

3. Map : Reading data in document - Tokenization - Stopword Filtering - Creating {Key, Value} : {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 a term dictionary file

5. Completion : Calculating a TF-iDF weight - Creating a vector representation file of all documents in collection

In order to identify 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 were summarized in Table 1.

Table 1: The summary of each experiment

Number	Purpose	Differentiated factors in implementation (1.Preparation, 2.Job Configuration, 3.Map, 4.Reduce, 5.Completion)
Case 1	Basic Test (Comparison Group)	1. Just copy input files from server's local file system to HDFS. 2. Processing unit for inputs : each file in 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. Make one input file in HDFS by concatenating all files in collection with tagging. 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. After reading one line, parse the tagged information for docID at first, and then use it to make pairs with tokenized terms until the end term of this line is processed. 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 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[17]. In Case 4, in order to extract the information of accurate docID(original file path before it was merged) 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 additional data as tagging information every time the new line is added. In our prototype, the tagging information only includes the docID, excepting other metadata.

4 Performance Evaluation

Document collection used in our experiments is the EDRM Enron Email Data Set v2[11]. 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 & SVM_Light by Thorsten Joachims at Cornell Univ[5].

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, does not affect our experiments at all.

4.2 Test Results

Table 2 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, there was no real effect of Hadoop initail job and 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 Related Work section that too many tasks in MapReduce cause the book-keeping overhead, so failing job is a natural outcome

Table 2: The time of job completion in each experiment

The 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

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 3 shows it was effective enough to improve the MapReduce performance on the other side.

Table 3: The Part of Hadoop Log Information in Case 3 : Non-Using vs. Using Snappy

	Counter	Non-use	Use
FileSystemCounters	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
	Map output materialized bytes / Reduce shuffle bytes	969,587,271	502,559,596

As you see in Table 3, Snappy optimizes the distribution of Map outputs for decreasing the number of times being read and written by I/O system. 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 such 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.

4.3 Addtional Considerations for the future DTPS

Currently the code for DTPS is a prototype and at the time of writing there are a number of items of missing functionality :

1. Current DTPS output is a vector representations of documents, the input files for using the

SVM_Light as an advanced search method. For the wide use of DTPS like a general search engine, designing the index file format based on its output should be a priority and the implementation of methods for query processing should be followed.

2. Current input file which was created by merging all documents in collection has a only one tagged information, docID. It means the rest metadata except document path and name cannot be directly used for search. Cosidering that the metadata can play an important part as evidence (for example, the date and time a document was written could be useful in a copyright case), this function will be positively necessary. To do this, the structured file format will be useful, so an XML-based approach is being planned.

3. After upgrading DTPS at the level of search engine, multiple users especially investigators will send a lot of queries at once for using a specific index file in HDFS. In order to avoid a single client using all available bandwidth, throttling techniques like a BlockTransferThrottler[2] originally provided by HDFS in Hadoop 0.20.2 will be applied to DTPS.

5 Conclusions

This paper described 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 is the development of text processing method for search in order to find relevant evidence more quickly and accurately from large-scale data. To do this, five experiments were conducted maniplating the code of MapReduce, the memory size of java heap and the type of input. As a result, we suggested that the best strategy of implementation for DTPS is the combination of making an integrated input file and compressing data processed in MapReduce task. On the guess that three different projects introduced in this paper is undesirable for processing the Big Data according to circumstances in digital investigation, our result clearly gives the direction on developing the advanced e-Discovery or digital forensic service. From now on, considering additional requirements of DTPS for using as the search engine, complete realization and research on the accuracy improvement of search will be our future work.

References

[1] 101tec, Inc. Katta. `http://katta.sourceforge.net/`, 2010.

[2] M. H. Butler and J. Rutherford. Distributed lucene : A distributed free text index for hadoop. Technical Report HPL-2008-64, Hewlett-Packard Development Company, L.P., May 2008.

[3] A. I. Cohen and E. G. Kalbaugh. *ESI Handbook : Sources, Technology and Process*. Aspen Publishers, October 2008.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[5] T. Joachims. Support Vector Machine, SVM_light. `http://www.cs.cornell.edu/people/tj/svm_light/`, 2008.

[6] J. Lee and S. Un. Digital forensics as a service: A case study of forensic indexed search. In *Proc. of the 2012 International Conference on ICT Convergence (ICTC'12), Jeju Island, Korea*, pages 499–503. IEEE, October 2012.

[2]This is a class to throttle the block transfers. It can be shared by multiple threads. The parameter bandwidthPerSec specifies the total bandwidth shared by threads. This class is thread safe.

[7] T. Lee, H. Kim, K. H. Rhee, and S. U. Shin. Design and implementation of e-discovery as a service based on cloud computing. *Computer Science and Information Systems*, 10(2):703–724, April 2013.

[8] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 0521865719.

[9] E. D. R. Model. EDRM Framework Guides. `http://edrm.net/resources/guides/edrm-framework-guides`, 2013.

[10] A. Tarantino. *Compliance Handbook(Technology, Finance, Environmental, and International Guidance and Best Practices)*. John Wiley & Sons Inc., December 2007.

[11] Text REtrieval Conference(TREC). Legal Track : Identification and download helpers for EDRM Enron v2 dataset. `http://trec-legal.umiacs.umd.edu/corpora/trec/legal10/`, 2010.

[12] The Apache Software Foundation. Apache Lucene Core. `http://lucene.apache.org/core/`, 2013.

[13] The Apache Software Foundation. Apache Nutch. `http://nutch.apache.org`, 2013.

[14] THE COMMITTEE ON THE JUDICIARY HOUSE OF REPRESENTATIVES. Federal rules of civil procedure. `http://www.uscourts.gov/uscourts/RulesAndPolicies/rules/2010%20Rules/Civil%20Procedure.pdf`, 2010.

[15] L. Volonino and I. J. Redpath. *e-Discovery For Dummies*. Wiley Publishing, Inc., November 2009.

[16] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, May 2012.

[17] Wikipedia. Snappy (software). `http://en.wikipedia.org/wiki/Snappy_(software)`, 2013.

Author Biography

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.

Hun Kim received his B.S. degree in Major of Computer and Multimedia Engineering from Pukyong National University, Busan, Korea in 2012. He is currently pursuing his master's degree in Department of Information Security, Graduate School, Pukyong National University. His research interests include digital forensics, e-Discovery, Cloud System and security.

Kyung-Hyune Rhee received his M.S. and Ph.D. degrees from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea in 1985 and 1992, respectively. He worked as a senior researcher in Electronic and Telecommunications Research Institute (ETRI), Daejon, 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 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/wireless network security and multimedia content security.