Private Search Over Big Data Leveraging Distributed File System and Parallel Processing

Ayse Selcuk, Cengiz Orencik and Erkay Savas

Faculty of Engineering and Natural Sciences
Sabanci University, Istanbul, Turkey
Email:{ayseselcuk, cengizo, erkays}@sabanciuniv.edu

***Abstract*—In this work, we identify the security and privacy problems associated with a certain Big Data application, namely secure keyword-based search over encrypted cloud data and emphasize the actual challenges and technical difficulties in the Big Data setting. More specifically, we provide definitions from which privacy requirements can be derived. In addition, we adapt an existing work on privacy-preserving keyword-based search method to the Big Data setting, in which, not only data is huge but also changing and accumulating very fast. Our proposal is scalable in the sense that it can leverage distributed file systems and parallel programming techniques such as the Hadoop Distributed File System (HDFS) and the MapReduce programming model, to work with very large data sets. We also propose a lazy idf-updating method that can efficiently handle the relevancy scores of the documents in a dynamically changing, large data set. We empirically show the efficiency and accuracy of the method through an extensive set of experiments on real data.**

Keywords–Cloud computing; Big Data; Keyword Search; Privacy; Hadoop.

I. Introduction

With the widespread use of the Internet and wireless technologies in recent years, the sheer volume of data being generated keeps increasing exponentially resulting in a sea of information that has no end in sight. Although the Internet is considered as the main source of the data, a considerable amount of data is also generated by other sources such as smart phones, surveillance cameras or aircraft, and their increasing use in everyday life. Utilizing these information sources, organizations collect terabytes and even petabytes of new data on a daily bases. However, the collected data is useless unless it is possible to analyze and understand the corresponding information.

The emergence of massive data sets and their incessant expansion and proliferation led to the term, the *Big Data*. Accurate analysis and processing of Big Data, which bring about new technological challenges, as well as concerns in areas such as privacy and ethics, can provide exceptionally invaluable information to users, companies, institutions, and in general to public benefit. The information harvested from Big Data has tremendous importance since it provides benefits such as cost reduction, efficiency improvement, risk reduction, better health care, and better decision making process. The technical challenges and difficulties in effective and efficient analysis of massive amount of collected data call for new processing methods [1], [2], leveraging the emergent parallel processing hardware and software technologies.

Although the tremendous benefits of big data are enthusiastically welcomed, the privacy issues still remain as a major concern. Most of the works in the literature, unfortunately, prefer to disregard the privacy issues due to efficiency concerns since efficiency and privacy protection are usually regarded as conflicting goals. This is true to certain extent due to technical challenges, which, however, should not deter the research to reconcile them in a framework, that allows *efficient privacy-preserving process of Big Data*.

A fundamental operation in any data set is to find data items containing a certain piece of information, which is often manifested by a set of keywords in a query, namely *keyword based search*. An important requirement of an effective search method over Big Data is the capability of *sorting* the matching items according to their relevancy to the keywords in queries. An efficient ranking method is particularly important in Big Data setting, since the number of matching data items will also be huge, if not filtered depending on their relevance levels.

In this paper, we generalize the privacy-preserving search method proposed in our previous work [3] and apply it in the Big Data setting. The previous version [3], which is sequentially implemented, was only capable of working with small data sets that have sizes of only a few thousand documents. In order to get more prominent and explicit results using massive data, we leverage the Hadoop framework [4] which is based on the distributed file systems and parallel programming techniques. For relevancy ordering, we use the well known tf-idf weighting metric and adjust it to dynamic Big Data. Unlike the work in [3], we assume the data set is dynamic, which is an essential property of Big Data. Therefore, we propose a method that we call "Lazy idf Update" which approximates the relevancy scores using the existing information and only updates the inverse document frequency (idf) scores of documents when the change rate in the data set is beyond a threshold. Our analysis demonstrates that the proposed method is an efficient and highly scalable privacy preserving search method that takes advantage of the HDFS [4] and the MapReduce programming paradigm.

The rest of this paper is organized as follows. In the next section (Section II), we briefly summarize the previous work in the literature. The properties of Big Data and the new technologies developed for the requirements of Big Data are summarized in Section III. In Section IV, we formalize the information that we hide in the protocol. The details of distributed file systems and the Hadoop framework are given in Section V. Section VI briefly summarizes the underlying search method of Orencik et al.[3]. The novel idf updating method for adjusting the tf-idf scoring for dynamically changing data set is explained in Section VII. In Section VIII, we discuss the results of the several experiments we applied on multi-node Hadoop setting. Section IX is devoted for the final remarks and conclusion.

II. Related Work

There are a number of works dealing with search over encrypted cloud data but most are not suitable for the requirements of Big Data. Most of the recent works are based on bilinear pairing [5]–[7]. However, computation costs of pairing based solutions are prohibitively high both on the server and on the user side. Therefore, pairing based solutions are generally not practical for Big Data applications.

Other than the bilinear pairing based methods, there are a number of hashing based solutions. Wang et al. [8] proposed a multi-keyword search scheme, which is secure under the random oracle model. This method uses a hash function to map keywords into a fixed length binary array. Cao et al. [9] proposed another multi-keyword search scheme that encodes the searchable database index into two binary matrices and uses inner product similarity during matching. This method is inefficient due to huge matrix operations and it is not suitable for ranking. Recently, Orencik et al. [3] proposed another efficient multi-keyword secure search method with ranking capability.

The requirements of processing Big Data led the big companies like Microsoft and Amazon to develop new technologies that can store and analyze large amounts of structured or unstructured data as distributed and parallel. Some of the most popular examples of these technologies are the Apache Hadoop project [4], Microsoft Azure [10] and Amazon Elastic Compute Cloud web service [11].

III. Challenges

As the name implies, the concept of Big Data is a massive dynamic set that contains a great variety of data types. There are several dimensions in Big Data that makes management a very challenging issue. The primary aspects of Big Data is best defined by its volume (amount of data), velocity (data change rate) and variety (range of data types) [12].

Unfortunately, standard off-the-shelf data mining and database management tools cannot capture or process these massive unstructured data sets within a tolerable elapsed time [13]. This led to the development of some new technologies for the requirements of Big Data.

In order to meet the scalability and reliability requirements, a new class of NoSQL based data storage technology referred as *Key-Value Store* [14]was developed and widely adopted.

This system utilizes associative arrays to store the key-value pairs on a distributed system. A key-value pair consists of a value and an index key that uniquely identifies that value. This allows distributing data and query load over many servers independently, thus achieving scalability. We also adapt the *key-value store* approach in the proposed method, where the details are explained in Section V.

IV. Privacy Requirements

In the literature, the privacy of the data analyzed by Big Data technologies is usually protected by anonymizing the data [15]. However, anonymization techniques are not sufficient to protect the data privacy. Although a number of searchable encryption and secure keyword search methods are proposed for the cloud data setting [5], [6], [9], none of them is suitable for Big Data.

A secure search method over Big Data should provide the following privacy requirements.

Definition 1. ***Query Privacy:*** *A secure search protocol has query privacy, if for all polynomial time adversaries A that, given two different set of search terms F_0 and F_1 and a query Q_b generated from the set F_b, where $b \in_R \{0,1\}$, the advantage of A in finding b is negligible.*

Intuitively, the query should not leak the information of the corresponding search terms.

Definition 2. ***Data Privacy:*** *A secure search protocol has data privacy, if the encrypted searchable data does not leak the content (i.e., features) of the documents.*

The search method we adapted [3] satisfies both privacy requirements. We do not repeat the proofs here and refer to the original work.

V. Distributed File Systems

It is not possible to process large amounts of data that are in the order of terabytes by using only a single server, due to the storage and computation power requirements. Therefore, we utilize the cloud computing services by software as a service (SaaS), which provide the use of shared computing hardware resources over a network on a pay-as-you-go basis [16]. Most of the cloud computing platforms use the Hadoop [4], which is an open-source distributed and paralleled framework. It provides easy and cost-effective processing solutions for vast amounts of data. The Hadoop framework is comprised of two main modules, which are the HDFS [17] for storing large amounts of data and accessing with high throughput and the MapReduce framework for distributed processing of large-scale data on commodity machines.

A. Hadoop HDFS

The HDFS is an open source file system that is inspired by the Google file system (GFS) [18]. The HDFS architecture runs on distributed clusters to manage massive data sets. It is a highly fault-tolerant system that can work on low-cost hardware. In addition, the HDFS enables high throughput access for application data and streaming access for file system data. The HDFS is based on a master/slave communication model that is composed of a single master node and multiple data (i.e. slave) nodes. There exists a unique node called the NameNode that runs on the master node. The master node manages the file system namespace to arrange the mapping between the files and the blocks and regulates the client access to the files [1].

B. Hadoop Mapreduce

Hadoop's MapReduce is based on Google's MapReduce algorithm [19]. The MapReduce programming model is derived from the Map and the Reduce functions which are used in functional programming beforehand. The MapReduce Programming model which processes massive data, provides large-scale computations for large clusters by dividing them into independent splits. The input data for the MapReduce is stored in the HDFS. The MapReduce utilizes the key-value pairs for distributing the input data to all the nodes in the cluster, in a parallel manner [20].

VI. Secure Search Method

The utilized privacy preserving search method is based on our previous work [3]. In this section, we briefly explain the method for completeness and refer the reader to [3] for the details. The search method is based on the minhashing technique [21]. Each document is represented by a constant length set called signature. During the similarity comparison, only the signatures are used and the underlying document feature sets are not revealed to the cloud. While this method cannot provide the exact similarity value, it can still provide a very accurate estimation. The signature of a document is defined as follows.

Definition 3. ***Minhash:*** *Let Δ be a finite set of elements, P be a permutation on Δ and $P[i]$ be the i^{th} element in the permutation P.* Minhash *of a set $D \subseteq \Delta$ under permutation P is defined as:*

$$h_P(D) = min(\{i \mid 1 \leq i \leq |\Delta| \ \wedge \ P[i] \in D\}). \quad (1)$$

For the signatures, λ different random permutations on Δ are used so the final signature of a document feature set D is:

$$Sig(D) = \{h_{P_1}(D), \dots, h_{P_\lambda}(D)\}, \quad (2)$$

where h_{P_j} is the *minhash* function under permutation P_j.

A. Index Generation

The index generation is an offline operation initiated by the data owner and creates the secure searchable index that is outsourced to the cloud. The searchable index generation process is based on the bucketization technique [22], [23], which is a well known method for data partitioning.

For each minhash function and corresponding output pair, a bucket is created with bucket identifier B_k^i (i.e., i^{th} minhash function produces output k). Each document identifier is distributed to λ different buckets according to the λ elements of the corresponding signature. In addition to the document identifiers, the corresponding relevancy scores (i.e., tf-idf value) are also added to the bucket content ($V_{B_k^i}$).

Note that, both the bucket identifiers (B_k^i) and the content vectors ($V_{B_k^i}$) are sensitive information that needs to be encrypted before outsourcing to the cloud. The secure searchable index $\mathcal{I}$ is the combination of the encrypted bucket identifiers and the corresponding encrypted content vectors.

B. Query Generation and Search

The query is generated in the same way as generating secure index entries. Given the set of keywords to be searched for, the query signature is generated by using the same minhash functions used in the index generation phase. The elements of the query signature are indeed the identifiers of the buckets that include the documents that contain the queried keywords. The bucket identifiers in the query signature are encrypted using the same secret keys used in index generation. The query is this set of encrypted bucket identifiers. Independent of the number of queried keywords, the query signature, hence the query itself has constant length, which is λ.

In the search phase, the cloud server receives the query and sends the requested encrypted content vectors to the user. The user then decrypts the vectors and ranks the document identifiers according to their relevancy with the query using the tf-idf scores. Finally, the user retrieves the encrypted documents with the highest relevancy scores from the server. Alternatively, the operation performed by the user can be handled by a trusted proxy, relieving the burden on the user.

VII. Relevancy Scoring

In the information retrieval setting, the results are required to be ordered according to their relevancy with the query. A commonly used scoring metric for information retrieval is the tf-idf weighting [24]. Intuitively, it measures the importance of a term within a document for a database collection. The tf-idf weighting uses both the term frequency (tf) and the inverse document frequency (idf) metrics. The term frequency of a term w is the the normalized number of times that w occurs in a document. The inverse document frequency measures the rarity of a term within the whole data set. The tf-idf of a term w in a document D is calculated as given in (3).

$$\text{tf-idf}_{w,D} = \text{tf}_{w,D} \times \text{idf}_w. \quad (3)$$

Generally, some tools are used for calculating the tf-idf weight such as the Rapid Miner [25], which is a popular text mining tool. Li and Guoyong [26] proposed an efficient method for calculating the tf-idf algorithm based on the Hadoop Framework. We use this algorithm to calculate the tf-idf weights in our test data sets.

A. Lazy idf Update

The Big Data necessitates high velocity in the data set which means new data is added continuously. The tf-idf metric uses the inverse document frequency (idf) (i.e., rarity within the data set) of each keyword. Let a document D contain k previously indexed terms. As this new document D is included to the data set, the scores of all the documents that contain any of those k terms should be updated since their idf values change. However, dynamically applying this change for each data item, added or removed from the data set, is not feasible. Hence, we propose a *lazy idf updating* method which aims to maintain the scores of existing documents as they are and only set a new score for the newly added items. Moreover, calculating the idf of each term of a newly added data item is still a costly operation that requires scanning the whole data set. In order to reduce the cost of scoring, we propose keeping the idf values of the terms separately. As new data elements are added, the idf values slightly change and the stored idf values will not exactly be correct. However, they still provide accurate estimates since the size of the existing data set is much larger than the size of the data elements added. In a timely bases (e.g., every 20 minutes), the whole data set is scanned and all the idf values are updated with the exact results.

Due to the privacy requirements, the server cannot see the actual documents but only stores the encrypted versions. It is not possible to calculate, neither the term frequencies, nor the inverse document frequencies from the encrypted data, therefore a trusted proxy should be used for updating the relevancy scores. Each new data item is first indexed and encrypted by the proxy and then uploaded to the server. Similarly, the idf value update operation is also done by the proxy. Therefore, the idf values that are separately stored are only kept in the trusted proxy. Since the idf update operation is performed by the proxy, the cloud server will be up and running during this period and the search operation can be done using the existing relevancy scores.

We assume that the size of the data set will be very large, hence the effect of the additional items on the idf values will be very limited. Note that, the term frequency (tf) part of the tf-idf score is calculated using only the document itself. Therefore, the change in the data set does not affect the tf values of the existing items. With this *lazy idf updating* method, very close estimates on the real tf-idf scores can be calculated in a very efficient way, hence it is suitable for the Big Data setting. The actual comparative results using a large, real data set is provided in Section VIII-C.

VIII. Experimental Results

In this section, we extensively analyze and demonstrate efficiency and effectiveness of the proposed method. The entire system is implemented by Java language using 64-bit Ubuntu 12.04 LTS operating system. In order to observe the benefits of distributed file systems, a multi-node Hadoop cluster is configured. The interface of Cloudera CDH4 with a three node (i.e., computer) cluster is utilized in the experiments. Two of the computers have an Intel Xeon CPU E5-1650 @ 3.5 GHz processor with 12 cores, 15.6 GB of main memory and the other computer has an Intel i7 @ 3.07 GHz processor with 8 cores and 15.7 GB of main memory.

In our experiments we used the Enron data set [27], which is a real data set that contains approximately $517,000$ email documents. Although the actual Enron data set is about 200 GB, we require a much smaller space as each document is represented by a single signature only, regardless of the size of the document.

A. Performance of the Method

In this section, we present the experiment results, where we measure the time spent for generating the secure searchable index and applying search operation.

The index generation time for $517,000$ documents, is given in Fig. 1 for different values of λ. The experiments demonstrate that the index generation, which is the most time consuming part of the method, can be done in only a few minutes. And the system can index about 2750 documents per second for $\lambda = 100$. Note that this operation is done only after the change in the data set, due to the documents added, exceeds a threshold. Moreover, since this operation is done by a trusted proxy, the cloud server can still continue to serve the incoming search requests, using the existing index.

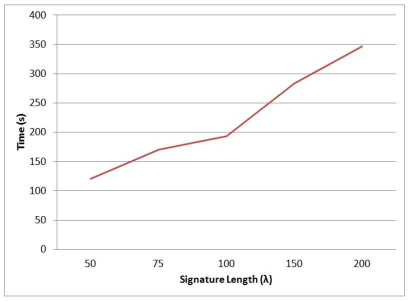

Figure 1. Index Generation Time as λ change

The search operation has two major parts. First the server fetches the content vectors of the queried buckets and sends them to the user. Then the user (or trusted proxy) decrypts those vectors and sorts the document identifiers according to the corresponding relevancy scores. Unfortunately, due to the distributed setting of the Hadoop file system, finding the queried buckets requires a search over all the created buckets. Fig. 2 demonstrates the average search time required both for the server and user sides, in the data set of size $517,000$ documents.

B. Accuracy of the Method

In the information retrieval community, two of the most common metrics for measuring the accuracy of a method are precision and recall. The metrics compare the expected and the actual results of the evaluated system. Precision measures the ratio of correctly found matches over the total number of returned matches. Similarly, recall measures the ratio of correctly found matches over the total number of expected results. Both precision and recall are real values between 0 and 1, where the higher the value the better the accuracy is.

In the case of a single term search, the proposed method guarantees all the matches that have non-zero relevancy scores, contain the searched term. Hence, retrieving all the items with non-zero scores satisfies perfect precision and recall. In the case of multiple keyword search, the matches with non-zero scores definitely contain at least one of the queried keywords

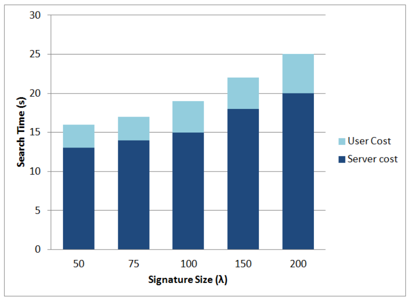

Figure 2. Search Time

but it may or may not contain all. We test the accuracy of the method for multi-term queries with $\lambda = 100$ (i.e., signature length) using the precision and recall metrics. The average precision and recall rates for a set of 20 queries with 2 and 3 keywords are given in Fig. 3 and 4, respectively. The retrieval ratio in the figures represents the ratio of the documents with nonzero scores that are considered as a relevant match with the query. The figures show that while the precision slightly decreases as retrieve ratio increases (i.e., more documents are considered as match), the recall increases. The retrieve ratio can be selected by the user according to the requirements of the application. The figures also show that the increase in the number of keywords decreases the precision but increases recall. The main reason of this is that, as the number of queried terms increases only very few documents contain all the queried terms which have a positive effect on recall. However, this also increases the documents with nonzero scores (i.e., contain at least one of the queried terms) which have a negative effect on precision.

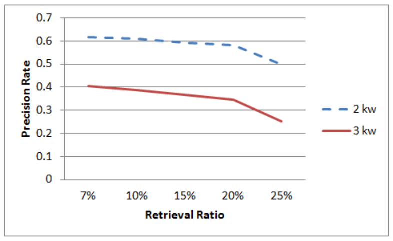

Figure 3. Average Precision Rate, $\lambda = 100$

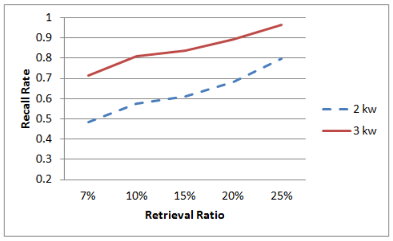

Figure 4. Average Recall Rate, $\lambda = 100$

Although the precision and recall metrics are very commonly used and very suitable for several problems such as conjunctive search and relational database search over structured data, they may not be very accurate for multi-keyword search over unstructured data. The main difference between search over structured and unstructured data is that, in the case of structured data, each field has an equal importance and the corresponding results should satisfy all the queried features. However, in the case of search over unstructured data, some of the queried features may be significantly more important than the others. For example, let a query has three features and a document contains only two of those features but with very high tf-idf scores. The precision and recall metrics will consider this document as a false match since it does not contain all the queried features, but in the case of Big Data we claim that this document is very relevant with the given query and should be considered as a match. It is important to note that, precision and recall metrics cannot consider the importance of the queried features in the compared document, hence may not perfectly measure the success rate of a search method over Big Data. Therefore, we also compare the output of the method with the ground truth. For calculating ground truth, the documents with top 50 scores in the data set are considered as the actual match results, where the complete tf-idf scores of the documents are used without any encryption. These actual results are then compared with the results evaluated by the system. The average precision and recall rates in comparison with the ground truth, for a set of 20 queries with 2 and 3 keywords and $\lambda = 100$, are given in Fig. 5 and 6, respectively. The figures show that, the actual accuracy of the method is quite promising when the tf-idf scores are considered in calculating the actual results, instead of the conjunctive (i.e., contain all terms) case.

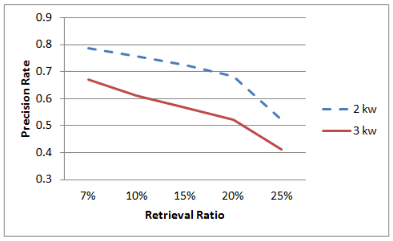

Figure 5. Average Precision Rate using Ground Truth, $\lambda = 100$

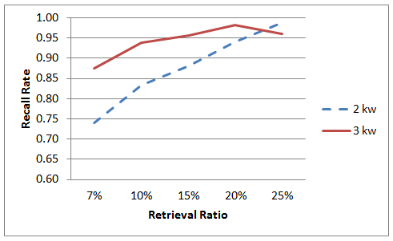

Figure 6. Average Recall Rate using Ground Truth, $\lambda = 100$

We also measure the effect of λ on accuracy. Fig. 7 and 8 show that an increase in λ has a positive effect on both precision and recall. However, increase in λ also linearly increases search and index generation times as shown in Section VIII-A and improvement in accuracy is very limited. Hence, an optimum value for λ should be set according to the properties of the data set used, which is set as 100 in our case.

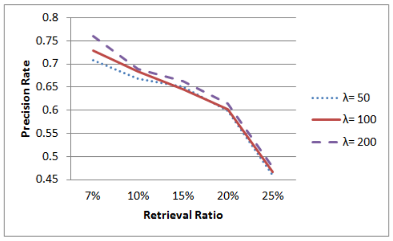

Figure 7. Average Precision Rate for different λ

C. Data Set Update

In Section VII-A, we propose a lazy update scheme that does not update the idf scores of the existing scores at each update but uses the existing scores as an approximation. In

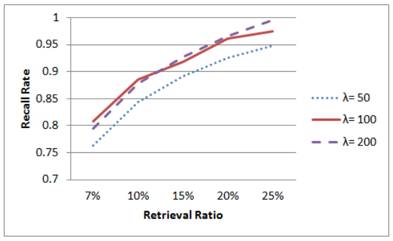

Figure 8. Average Recall Rate for different λ

this subsection, we provide the change rate of the idf due to update in the data set. We calculate the average idf scores of a data set of size $400,000$ documents while adding a new set of documents of size $10,000$. As Table I indicates, the effect of adding new documents is very low especially if the data set size is large, hence the lazy update does not reduce accuracy.

TABLE I. AVERAGE IDF VALUES

# documents	400,000	410,000	420,000	430,000
avg idf	2.26678	2.26716	2.26713	2.26717

Inserting index entries for documents added first requires calculating the corresponding signatures by a trusted proxy and than updating the encrypted bucket content vectors accordingly. We tested the update times for bulk insertions for 1000, 5000 and 10000 documents and the whole update operations are calculated as 52.5, 59.5 and 67 seconds, respectively. This shows that the update operation should be done for large sets of documents which is also suitable for the Big Data setting.

IX. CONCLUSIONS

In this work, we addressed the problem of applying an existing privacy-preserving search method for the case of Big Data. We utilized the search method of Orencik et al. [3] as the underlying search method and applied it for the HDFS and the MapReduce programming model. We implemented the entire system and tested for a three-node Hadoop with the Enron email data set and demonstrate the effectiveness and scalability of the system. We also proposed a *lazy idf update* method that can be used for dynamically changing large data sets and provide extensive results using a large real data set.

In the light of the promising results, we believe this method will increase the applicability of privacy preserving search over Big Data.

ACKNOWLEDGMENT

Ayse Selcuk was supported by ARGELA, under Grant Number 3014-07. Dr. Orencik and Dr. Savas were supported by TUBITAK under Grant Number 113E537.

REFERENCES

[1] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-hadoop: Mapreduce across distributed data centers for data-intensive computing," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 739–750, 2013.

[2] L. Wang, M. Kunze, J. Tao, and G. von Laszewski, "Towards building a cloud for scientific applications," *Advances in Engineering Software*, vol. 42, no. 9, pp. 714–722, 2011.

[3] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *CLOUD 2013*, pp. 390–398, IEEE, 2013.

[4] http://hadoop.apache.org/core/, 2009. [accessed Nov 2014].

[5] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *J. Netw. Comput. Appl.*, vol. 34, pp. 262–267, Jan. 2011.

[6] Z. Chen, C. Wu, D. Wang, and S. Li, "Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor," in *PAISI*, pp. 176–189, 2012.

[7] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology, CRYPTO 2013*, vol. 8042 of *Lecture Notes in Computer Science*, pp. 353–373, 2013.

[8] P. Wang, H. Wang, and J. Pieprzyk, "An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data," in *Information Security Applications*, Lecture Notes in Computer Science, pp. 145–159, Springer, 2009.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *IEEE INFOCOM*, 2011.

[10] https://azure.microsoft.com/, 2014. [accessed Jan 2015].

[11] http://aws.amazon.com/ec2/, 2014. [accessed Jan 2015].

[12] D. Laney, "3d data management: Controling data volume, velocity and variety," 2001.

[13] C. Snijders, U. Matzat, and U. Reips, "Big data: Big gaps of knowledge in the field of internet," in *International Journal of Internet Science*, 2012.

[14] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pp. 205–220, ACM, 2007.

[15] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, pp. 363–373, Feb. 2014.

[16] Amazon Web Services, "What is cloud computing." http://aws.amazon.com/what-is-cloud-computing/. [accessed Dec 2014].

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1–10, IEEE, 2010.

[18] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, pp. 29–43, ACM, 2003.

[19] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[20] B. T. Rao and L. Reddy, "Survey on improved scheduling in hadoop mapreduce in cloud environments.," *International Journal of Computer Applications*, vol. 34, 2011.

[21] A. Rajaraman and D. Ullman, Jeffrey, *Mining of massive datasets*. Cambridge University Press, 2011.

[22] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pp. 216–227, ACM, 2002.

[23] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, pp. 333–358, June 2012.

[24] H. S. Christopher D. Manning, Prabhakar Raghavan, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[25] https://rapidminer.com/. [accessed Jan 2015].

[26] B. Li and Y. Guoyong, "Improvement of tf-idf algorithm based on hadoop framework," 2012.

[27] "Enron email dataset." http://www.cs.cmu.edu/enron, Jan. 2012. [accessed Nov 2014].