iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** VII **Month of publication:** July 2017

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

Review of Lucene Indexing Algorithm on Public Cloud

Ms. Deepali Dattatray Rane[1] Mrs. Shraddha T. Shelar[2]
[1, 2]*Assistant Professor, Department. of Information Technology. DYPCOE, Akurdi.*

Abstract: In today's world users are generating huge amount of data therefore they have to outsource it to the public cloud. Cloud server is not trusted; data security is the main concern for data owners. To preserve the privacy as well as security of documents, it should get encrypted before outsourcing to the cloud. As users documents get uploaded to public cloud in encrypted format, they lose physical possession of their private data. Users should have flexibility to access their data from anywhere at any time. Lucene indexing algorithm will fetch top ranked documents from cloud database matching specified keywords specified in multi-keyword search query.
Keywords: Privacy , Multi-Keyword Query, Ranking, Indexing, Lucene

I. INTRODUCTION

Cloud computing means storing and accessing data and programs over the Internet instead of your computer's hard drive. The cloud is just a metaphor for the Internet. There are three types of cloud: public cloud, private cloud and hybrid cloud. In today's era many users want to have a secure storage for their data, it this situation cloud comes into picture. Users are motivated to outsource their personal as well as professional data to public cloud, but they want their data to be safe at cloud side because as they outsource it, they lose physical possession of their data. Public cloud storage is not trusted storage. To protect data confidentiality and unauthorized access to the cloud data, owners are motivated to encrypt the data before outsourcing to the cloud. As user is going to store encrypted data at cloud side, traditional searching will not be effective.
To meet the effective searching on the encrypted cloud data, Multi-Keyword query should be formed and fired so as to get the top relevant data of user interest. Numbers of public cloud service providers are there like Amazon EC2, Google drive, Rackspace, Jelastic etc which will prove users to freely store their data on cloud. Indexing is a common operation in web search engines. Lucene indexing algorithm has been used for building index of keywords. It is modified version of vector space model and the index created is inverted index. Analysis has been made to check whether appropriate segments are created or not as documents get uploaded and index got created for each document. Based on score, searching will prioritize the results.

II. RELATED WORK

Previous systems have implemented various searching schemes which have utilized bilinear maps and also which are based on public key encryption technique. When searching for a particular document, either single keyword searching or single user searching has been developed. Users are sending the query consisting of keywords, hence unable to hide the search pattern. Some schemes are developed in which user must have knowledge about all the valid keywords and their respective positions as mandatory information so as to generate a query.
Disadvantages of Existing System can be ,

A. Single-keyword search without ranking
B. Boolean keyword searching without ranking
C. Single-keyword search with ranking
D. Rarely sorting of the results i.e. no index creation and ranking
E. Single User search

Vector space model procedure has been divided into three parts namely document indexing, weighting of the indexed terms to enhance retrieval of document relevant to the user and third one is ranking the document with respect to the query according to a similarity measure.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VII, July 2017- Available at www.ijraset.com

III. PROPOSED ALGORITHM

A. System development

Objective of the proposed system is to extract the top relevant data similar to users query as well as to enhance experience of users while searching. Single keyword searching technique generates unwanted traffic of data which is not that much related to users query. Hence we need to develop robust system which will provide efficient results with multi-keyword searching.

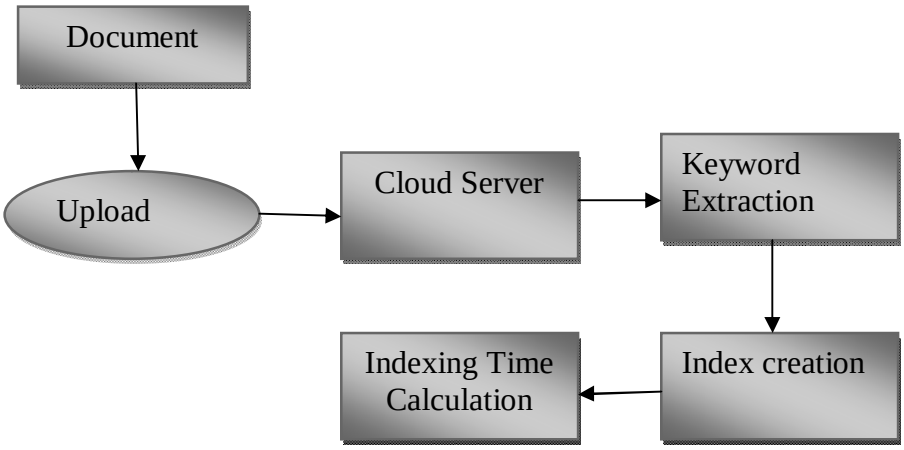


Fig 1 A): Architecture of index creation

Proposed system fetches the keywords from the document and builds index of keywords at server side. Keyword Index will help user to match the keywords from query to the documents in order to fetch top *k* relevant data. When data owner decides to upload his personal documents to the cloud, first the keywords from the documents will get fetched and index will get created. As documents get added and deleted accordingly the index will get updated. Keywords first get hashed and hashed value of keywords will get stored in index. As documents will get added to cloud space, index segments will get created and according to merge factor, index will get updated. Lucene indexing algorithm is used for creating index of documents.

B. Lucene indexing

When individual wants to upload his/her official or personal documents on the cloud server, keywords get extracted from that documents and index will be generated [1], [4]. Hashed values of keywords are stored as index. When document will get added and removed from cloud storage, index will get restructured accordingly.

The basic motivation behind this module is to fetch the keywords from the document and build index of keywords at server side. Keyword Index will help user to match the keywords from query to the documents in order to fetch top *k* relevant data. When data owner decides to upload his personal documents to the cloud, first the keywords from the documents will get fetched and index will get created. Encrypted document and secret key will be uploaded to the server along with the index created from the keywords. As documents get added and deleted accordingly the index will get updated. Keywords first get hashed and hashed value of keywords will get stored in index. As documents will get added to cloud space, index segments will get created and according to merge factor, index will get updated. For creating index of keywords fetched from document, Lucene indexing algorithm has been used. Lucene is full-text search engine architecture, provides a complete query and indexing engine. Lucene has several advantages like an index is file format and application platform independent. It provides search over documents. It manages an index over dynamic collection of documents and provides very rapid updates to the index as documents are added to and deleted from the collection. An index may store a heterogeneous set of documents with any number of different fields that may vary by documents in arbitrary ways. The flexibility allows Lucene API to be independent of the file format. Text from pdf, Html, Microsoft word and openDocument as well as many others (except images) can all be indexed as their textual information can be extracted. Lucene Indexing algorithm is modified vector space model implementation and index created is **inverted index**. To index a document, we have to first scan it to produce list of Postings**.** Posting describes occurrences of word in a document. They generally include word, document ID and location or frequency of the word within document. Postings is a tuple of the form <word, document-id>, a set of documents will

yield a list of postings sorted by document ID. IBM, Technorati, Wikipedia, Internet Archive, LinkedIn, Hewlett Packard, Microsoft etc are the companies which are using Lucene for indexing purpose.

1) Indexing Process

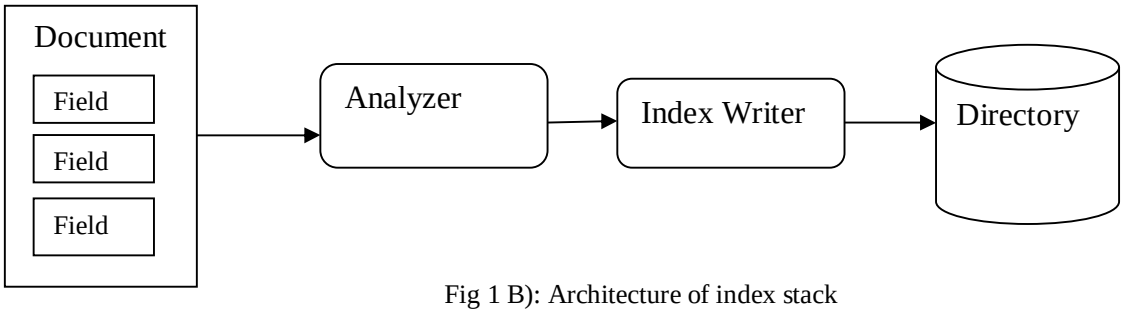


Fig 1 B): Architecture of index stack

User add Document(s) containing Field(s) to IndexWriter which analyzes theDocument(s) using the Analyzer and then creates/open/edit indexes as required and store/update them in a *Directory*. Index Writer is used to update or create indexes. It is not used to read indexes. An index contains a sequence of documents.

A document is a sequence of fields.

A field is a named sequence of terms.

A term is a string.

The index stores statistics about terms in order to make term-based search more efficient. Lucene's index falls into the family of indexes known as an *inverted index*. This is because it can list, for a term, the documents that contain it. Lucene indexes may be composed of multiple sub-indexes, or *segments*. Each segment is a fully independent index, which could be searched separately. Indexes evolve by:

a) Creating new segments for newly added documents.

b) Merging existing segments.

Searches may involve multiple segments and/or multiple indexes, each index potentially composed of a set of segments. Internally, Lucene refers to documents by an integer document number. The first document added to an index is numbered zero, and each subsequent document added gets a number one greater than the previous. Index directory contains .GEN file and .CFS files.

Supporting full-text search using Lucene indexing executes two steps:

c) Generating a lucence index on the documents and on database objects and

d) Parsing the user query and looking up the prebuilt index to answer the query.

2) Lucene Indexing Algorithm

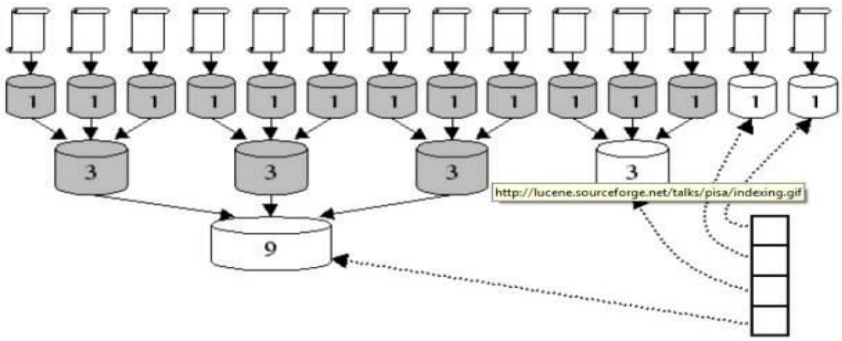


Fig 2: Working of Lucene indexing

a) Two basic algorithm

i) Make an Index for single Document.

ii) Merge set of Indices.

b) Lucene Incremental Algorithm

i) Maintain stack of segment indices.

ii) Create index for each incoming document

iii) Push new indexes onto the stack

iv) Let b=10 be the merge factor, M=∞

```
For (size=1; size<M; size*=b)
  {
     If (there are b indexes with size docs on top of stack)
     {
       Pop them off the stack; Merge them into a single stack; Push the merged index onto the stack;
       }
       Else
       {Break;
       }
  }
```

3) Lucene Tikka : The Apache Tika toolkit detects and extracts metadata and text from over a thousand different file types (such as PPT, XLS, and PDF). All of these file types can be parsed through a single interface, making Tika useful for search engine indexing. Apache Tika is a library that provides a flexible and robust set of interfaces that can be used in any context where metadata analysis and structured text extraction is needed. The key component of Apache Tika is the **Parser interface** because it hides the complexity of different file formats while providing a simple and powerful mechanism to extract structured text content and metadata from all sorts of documents. Using Tika, one can develop a universal type detector and content extractor to extract both structured text as well as metadata from different types of documents such as spreadsheets, text documents, images, PDFs and even multimedia input formats to a certain extent.

a) Functionalities of Tika

i) Document type detection

ii) Content extraction

iii) Metadata extraction

iv) Language detection

b) Apache tika supports following document formats :

i) HyperText Markup Language

ii) XML and derived formats

iii) Microsoft Office document formats

iv) OpenDocument Format

v) Portable Document Format

vi) Electronic Publication Format

vii) Rich Text Format

viii) Compression and packaging formats

ix) Text formats

x) Audio formats

xi) Image formats

xii) Video formats

xiii) Java class files and archives

xiv) The mbox format

4) *Retrieval of top ranked data:* As single keyword search in cloud will results in too much network congestion. To avoid network traffic, user can fire a search query consisting of multiple keywords. As user documents are stored at cloud side, when they want to cached these documents at their own workspace they build a search query consisting of multiple keywords.

Following are the key contributors used for building search engine,

a) *Document* - The Document represents a document in Lucene. We index Document objects and get Document objects back when we do a search.
b) *Field* - The Field represents a section of a Document. The Field object will contain a name for the section and the actual data.
c) *Analyser* - The Analyzer is an abstract class that is used to provide an interface that will take a Document and turn it into tokens that can be indexed. There are several useful implementations but the most commonly used is the **StandardAnalyzer** class.
d) *IndexWriter* - The IndexWriter is used to create and maintain indexes.
e) *IndexSearcher* - The IndexSearcher is used to search through an index.
f) *QueryParser* - The QueryParser class is used to build a parser that can search through an index.
g) *Query* - The Query is an abstract class that contains the search criteria created by the QueryParser.

Fig.3.Searching in cloud

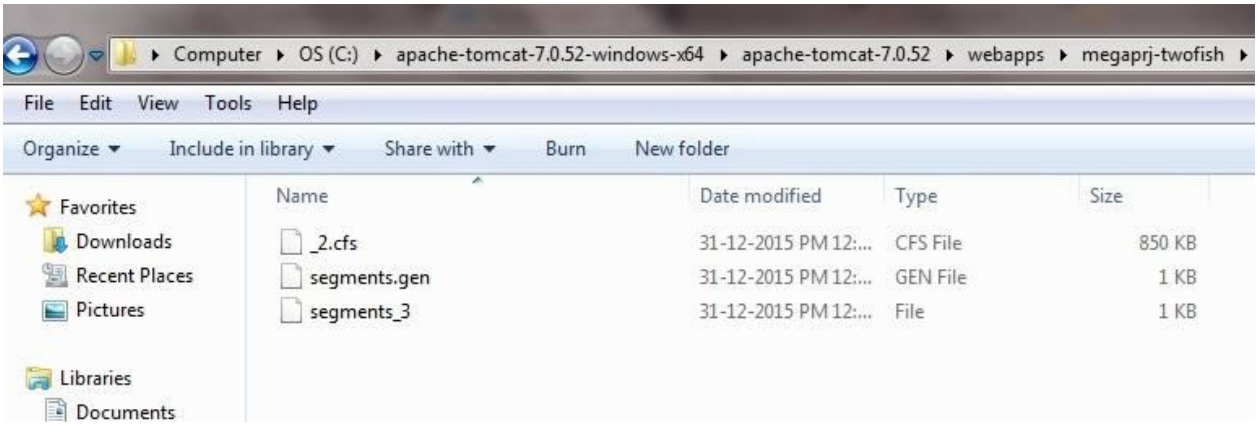


Fig.4 Index Construction

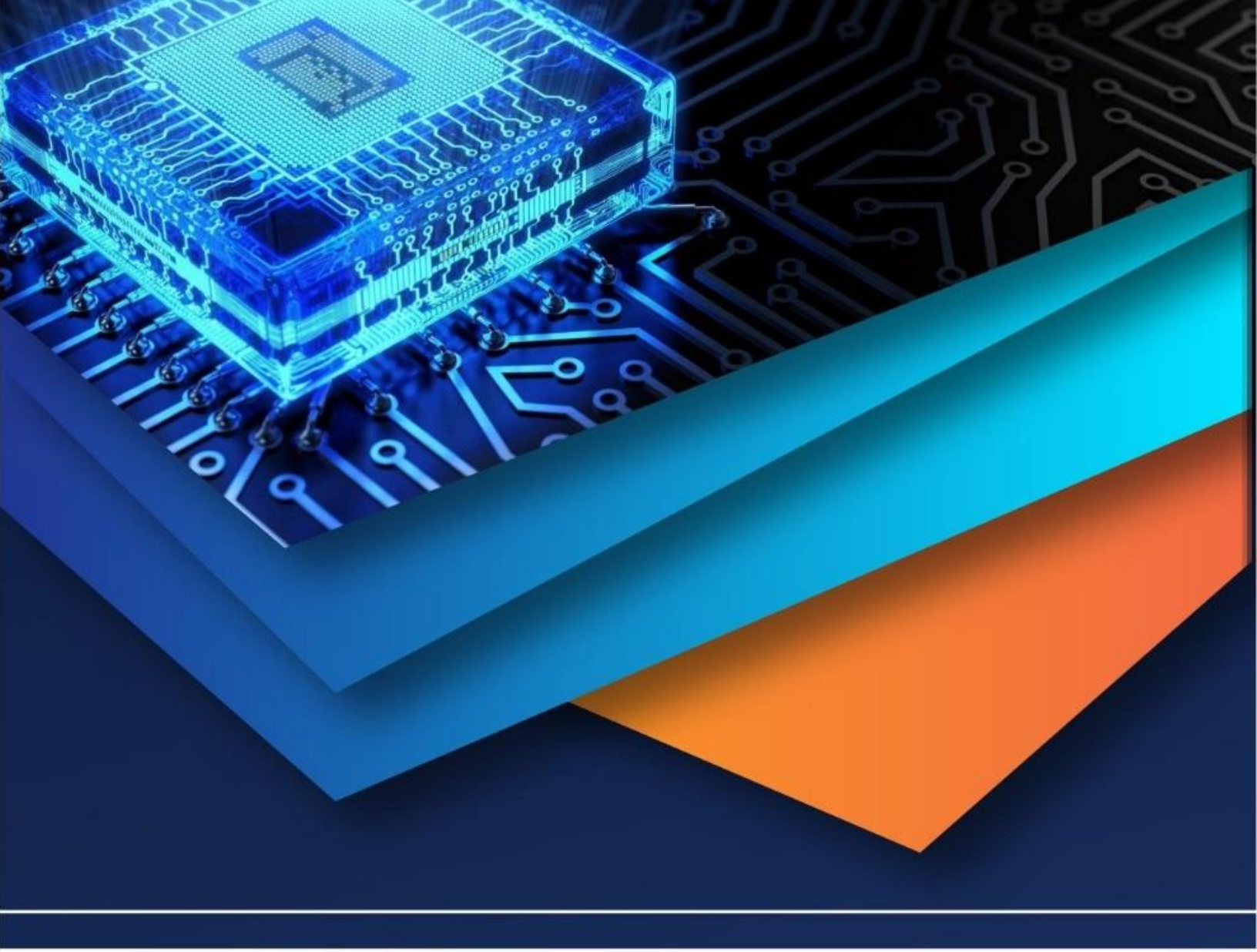
IV. CONCLUSION

Proposed paper focused on how the user can effectively retrieve their private documents from cloud while preserving privacy of their documents whenever and wherever by sending a query consisting of multiple keywords. In response to the users query, system will match the keywords from query to the documents using “keyword-matching principle”. Top ranked documents will get fetched which will consist of keywords specified by user in query. Checking the rank of the retrieved document can be done by calculating how many docs contains the specified keywords how many times.

REFERENCES

[1] Miss Deepali D.Rane, Dr.V.R.Ghorpade, “Multi-User Multi keyword Privacy Preserving Ranked Based Search over Encryted Cloud Data”, International Conference on Pervasive Computing (ICPC).

[2] Ning Cao, Cong Wang, Ming Li, kuiren, Wenjing Lou , "Privacy Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data" IEEE Transactions on parallel and Distributed Systems, Vol. 25,January 2014
[3] Ayad Ibrahim, Hai Jin, Ali A.Yassin, deqingzou, "Secure Rank-Ordered Search of Multi-Keyword Trapdoor over Encrypted Cloud Data" IEEE Asia-Pacific Services Computing Conference 2012
[4] Yanjiang Yang, "Towards Multi-User Private Keyword Search for Cloud Computing" IEEE 4th International Conference on Cloud Computing. 2011.
[5] Qin Liu, Guojun Wag, Jie Wu, "An Efficient Privacy Preserving Keyword Search Scheme in Cloud Computing" IEEE International Conference on Computational Science and Engineering, 2009.
[6] Twofish : A 128 Bit Block Cipher by Bruce Schneier, John Kelseyy, Doug Whitingz David, Wagnerx Chris Hall .
[7] "Analysis of AES and Twofish Encryption Schemes" IEEE Transaction 2011.
[8] Yong Zhang, Jian-lin Li, "Research and Improvement of Search Engine Based on Lucene" IEEE Transaction 2009.
[9] QIAN Liping, WANG Lidong, "An Evaluation of Lucene for Keywords Search in Large-scale Short Text Storage" IEEE Transaction 2010.
[10] Govind S.Pole, Madhuri Potey, "A Highly Efficient Distributed Indexing system based on large cluster of commodity machines" IEEE Transaction 2012.
[11] https://tika.apache.org/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY