Search Time Comparison between Structured and Semi-Structured Data over Inverted Index

B.Usharani
Dept. Of CSE,KLUniversity,INDIA

Abstract:

Data mining means mining knowledge from the data. Rapid growth of information on the web makes users turn to search engines. Users use keyword search to retrieve data by typing the keywords as search queries. The keyword search systems use an inverted index that maps each word in the index to a list of IDs of documents. An inverted index contains full positional information about all word occurrences in the collection. Inverted index is used to index the documents and to access the documents according to a set of keywords efficiently. Inverted Index is the backbone of all Search Engines. Search in inverted index is efficient than search in the normal documents. In this paper a time comparison is made to search in inverted index between structured data and semi- structured data.

Keywords: **inverted index, keyword search, query processing, structured, semi-structured, comparison times etc**

I. INTRODUCTION

Keyword search is the mechanism used for information discovery and retrieval. Keyword search is an efficient way to access the data Keyword search is a simple search model that can be issued by writing a list of keywords. Keyword based search enables users to easily access databases without the need either to learn a structured query language or to study complex data schemas. The essential task in keyword search is to display query results which automatically gather relevant information that is generally fragmented and scattered across multiple places.

The inverted index data structure is a key component of keyword search. The index for each term can be sorted in order of allowing ranked retrieval documents. Search engines use the traditional docid ordering, where each posting list is ordered by ascending document id, which permits efficient retrieval. The compression of the doc-id ordering of posting lists is the focus of our work. The proposed system deals with the document id(doc-id) and term frequency (tf) compression.

A. Index

An index is an alphabetically arranged list of words. Search engine index size is hundreds of millions of web pages. Search engines have to answer tens of millions of queries every day. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours. The additional computer storage required to store the index, as well as the considerable increase in the time required for an update to take place, are traded off for the time saved during information retrieval.[6]

The first Google Index in 1998 already has 26 million pages, and by 2000 the Google index reached the one billion mark.[7]

B. Inverted Index

The main objective of an inverted index is to provide fast text search. An inverted index(postings file or inverted file) is a data structure used for document retrieval. Inverted index have two variants :1)**record level inverted index**(inverted file index or inverted file) – tells you which document contains that particular word.2)**word level inverted index**(fully inverted index or inverted list)—tells you both which document contains that particular word and the place(position) of the word in that particular document. This can be represented as (document-id**;**no:of occurrences,[position]).For example

D_0– {this is to test a test}

Positions: 0 1 2 3 4 5

D_1—{this is testing to test application}

D_2—{this is a testing application}

The words would store in the inverted index like this:

Table 1

Inverted Index table

words	**Inverted file ndex**	**inverted list**
a	{0,2}	{(D_0;1,[4])(D_2;1,[2])}
application	{1,2}	{(D_1;1,[5])(D_2;1,[4])}
is	{0,1,2}	{(D_0;1,[1])(D_1;1,[1])(D_2;1,[1])}
test	{0,1}	{(D_0;2,[3,5])(D_1;1,[4])}
testing	{1,2}	{(D_1;1,[2])(D_2;1,[3])}
this	{0,1,2}	{(D_0;1,[0])(D_1;1,[0])(D_2;1,[0])}
to	{0,1}	{(D_0;1,[2])(D_1;1,[3])}

C. Semi-Structured Data

"Semi-structured data" refers to the data that has the structure. There is no strict formatting rule for the semi structured data. Semi-Structured is a structured data, but it is not organized in a rational model, like a table.

XML" is an example of a language for representing the semi-structured data.

The XML standard was first published by the W3C in Feb 1998.

Eg.xml file movies.xml

<?xml version="1.0" encoding="UTF-8"?>

<information>

<tuple>

<mid>m1 </mid>

<moviename> toy story (1995)</moviename>

<generes>comedy drama </generes>

<url> http://us.imdb.com/M/title-exact?Toy%20Story%20</url>

</tuple>

…….

</information>

In the above example, mid, movie name, generes, url contain the text whereas tuple and information contain the other elements.

To lower the weight of the terms with the high document frequency

$$idf_t = log\frac{N}{df_t} \quad (1)$$

Where n is the total no: of documents in the collection and df_t is a document frequency for a given term.

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \quad (2)$$

Replace term frequency tf with word frequency wf then the above (1) and (2),the corresponding can be written as:

$$wf_{t,d} = \begin{cases} 1 + log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$wf\text{-}idf_{t,d} = wf_{t,d} \times idf_t$$

The more weight schemes are:

Table 2

Term Frequency vs document frequency

Term frequency		Document frequency	
n (natural)	$tf_{t,d}$	n (no)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(ave_{t \in d}(tf_{t,d}))}$		

Recall

Recall is the proportion of relevant results retrieved in top-k results.

$$R_k = \frac{\text{number of relevant results retrieved}}{\text{total number of relevant results}}$$

Precision

Precision is the fraction of results retrieved that are relevant to a query.

$$P_k = \frac{\text{number of relevant results retrieved}}{\text{total number of retrieved results}}$$

II. INVERTED INDEX CONSTRUCTION CONSTRUCTION

Index provides a fast access to a subset of data records in data base. Every word in a document is a search word. The index can be accessed through the terms(words).Inverted lists are the most common indexing technique.

A. Database

Table 3

Movie table

mid	**movie name**	**generes**
m1	toy story (1995)	animation childres comedy
m2	jumanji (1995)	adventure children fantasy
m3	grumpier old men (1995)	comedy romance
m4	waiting to exhale (1995)	comedy drama
m5	father of the bride partII(1995)	comedy
m6	heat (1995)	action crime thriller
m7	sabrina (1995)	comedy romance

Table 4

User table

UID	UNAME
u1	john
u2	mike
u3	peter
u4	jimmy
u5	david
u6	richard

Table 5

Rating table

uid	mid
u1	m1
u1	m3
u2	m4
u3	m2
u3	m3
u4	m5
u4	m7
u5	m6
u6	m2

B. Tuple Unit Generator

The idea of tuple units is taken from [30].The tuple units (TU) is a set of tuples linked through primary or foreign keys and stores all the information extracted from the tuples. The Tuple Unit stores the relations between the tuples and contains keywords that can be used in a query[8] .[8][1][2][3][4]..For generating the tuple units the rating table information is used. The tuple unit consists of redundant information.

Eg .

Tu1 =

UID	Name	MID	Movie_Name	Generes	UID	Name
u1	Jhon	m1	Toy Story (1995)	Animation|Children's|Comedy		
		m3	Grumpier Old Men (1995)	Comedy|Romance	u3	Peter

Tu3 =

UID	Name	MID	Movie_Name	Generes	UID	Name
u3	Peter	m2	Jumanji (1995)	Adventure|Children's|Fantasy	u6	Richard
		m3	Grumpier Old Men (1995)	Comedy|Romance	u1	Jhon

Fig. 1 Tuple units for u1 and u3

The above table is constructed using the rating table. The user u1 votes for the movies m1 and m3.Again the movie m3 is voted by the user u3.The movie m1 is not voted by any other user. The tuple units are generated using the rating table. The tuple units for the whole database are given below:

The Tuple Unit-based method has the subsequent features:

- Tuple Units are effective to respond keyword queries as they implicitly capture the structural relationships and can correspond to an integral information unit.
- The relationships between tuples connected through primary/foreign keys can be recognized and indexed.
- The number of Tuple Units will not be large, which is not larger than the total tuples in the original database.

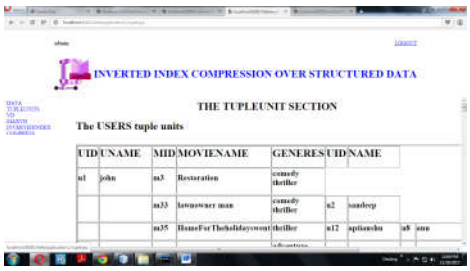

Fig .2 The tuple units for the users table

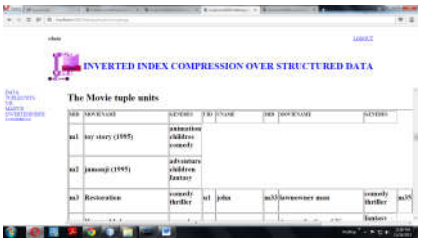

Fig 3: The tuple units for the movie table

Once all the tuple units have been created, the next step is to transform these tuple units into Virtual Documents. In this way, we will decrease redundant information in the tuple units and progress the indexing process.

The Tuple Unit-based method has the following features:

- TUs are effective to answer keyword queries as they implicitly capture the structural relationships and can represent an integral information unit.

- The relationships between tuples connected through primary/foreign keys can be identified and indexed.
- The number of TUs will not be large, which is not larger than the total tuples in the underlying database.[8]

The output of the tuple unit becomes the input to the virtual document generator.

C .Virtual Document Generator

The term Virtual Document (VD) was proposed by Su et al. [30] .The final set of tuple units created becomes the input for the virtual document generator. The VD generator is responsible for creating the virtual document (VD) by splitting the TU, detecting and minimizing redundant information in order to generate a set of unique and reduced VDs.[8] [1][2][3][4]..The Virtual documents removes the redundancy that is existed in the tuple units by splitting the tuple units into virtual documents.

Fig. 4 Virtual Documents Construction

One or more tuple units are generated from each tuple unit. In the above example the virtual documents are generated from tuple unit TU1.The virtual documents for the tuple unit TU1 are{{u1,m1,m3},{u1,m1,m3,u3}}.

The virtual documents for the whole database are given below:

The idea is to reduce the size of VDs by avoiding redundant.information. The new VD only contains the textual attributes from the Tuple Unit. The VDs are the input to the Indexing process.A Virtual Document (VD) is a document that is generated from a Tuple Unit

subsequent features

- It is most compact than a Tuple Unit.
- It eliminates duplicated information.
- It must be unique

It is possible to generate one or more VDs from one Tuple Unit. Since some generated Virtual Documents (VDs) could include the same information nodes but in different order, it is necessary to detect duplicate VDs

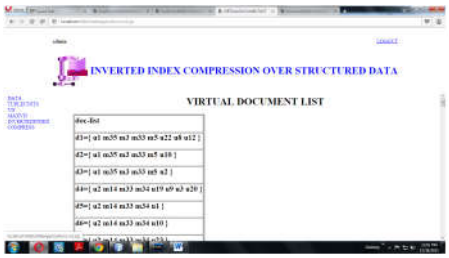

Fig 5:virtual document list

D. MaximalVirtualDocument(MVD)

Max virtual document Definition: Given a virtual document $vd_i \varepsilon VD$,vd_i is called Maximal virtual document if there is no other virtual document $vd_j \mid vd_j \varepsilon VD \wedge vd_i \not\subset vd_j$.

Since some generated Virtual Documents (VDs) could include the same information but in different order, it is necessary to detect duplicate VDs.For example Let's take the Virtual Documents Vd1 and Vd2 . Vd2 is considered a MVD because all the information that belong to Vd1 are contained in Vd2, hence Vd1 is eliminated. We identify similar cases in order to avoid duplicity,producing only a set of MVDs.

Consider this example:

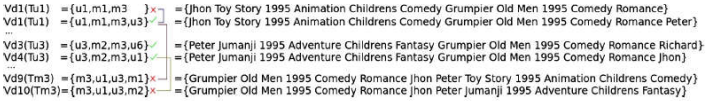

Fig .6 Detection of Maximal Virtual Documents

In the above example the virtual document vd1 i.e{u1,m1,m3} is included in the virtual document vd9 i.e {m3,u1,u3,m1}.The order varies but the contents of vd9 contains vd1.So,there is no need to maintain vd1.Maintaing the vd1 is nothing but wastage of memory space. So, remove the virtual document vd1. The order of the id's are not considered .Suppose one virtual document D1 consists of contents{a,b,c,d}.Another

virtual document D2 consists of contents {b,c,d,a}.The documents D1 and D2 consists of same contents but the order of contents is different. So, maintain the two documents in the memory is wastage of memory space. No need to maintain two documents. One document is sufficient from those two documents i.e. from D1 or D2.

By checking all the virtual documents, remove the duplicate documents. The resultant set is the set of Maximal virtual document set. The final MVD contains the unique virtual documents. The final MVD for the whole database is given below:

The MVD can be produced by avoiding the duplicity in the VD.

Fig . 7 maximal virtual document list

E . Inverted Index

The inverted index can be constructed from the keywords contained in the MVD .Every Keyword points to a list of information where the keyword exists in the particular document and the count of that particular keyword in that document. The index entries are the keywords contained in the Virtual Documents (VDs). Each entry (keyword) points to a list of VDs where the keyword is contained,and every VD includes an associated score that indicates the weight (relevance) that this keyword has in the VD. The VDs in every list are sorted in descending order

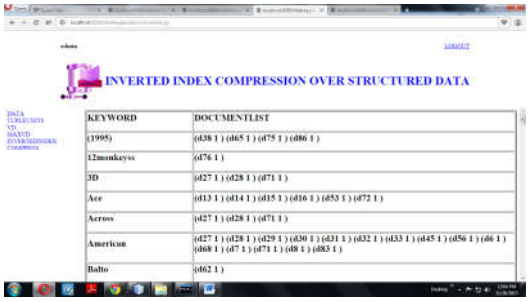

Fig .8 Inverted index list

The inverted index can be constructed from the keywords contained in the MVD .Every Keyword points to a list of information where the keyword exists in the particular document and the count of that particular keyword in that document. The index entries are the keywords contained in the Virtual Documents (VDs). Each entry (keyword) points to a list of VDs where the keyword is contained,and every VD includes an associated score that indicates the weight (relevance) that this keyword has in the VD. The VDs in every list are sorted in descending order.The inverted index can be constructed by using the equations (1) and (2).

Table 6

Inverted index list

Term	Docnumber	Freq
ambitious	2	1
be	2	1
brutus	1	1
capitol	2	1
caesar	1	1
did	1	1
enact	2	2
hath	1	1
I	1	1
i'	2	1
it	1	2
julius	1	1
killed	2	1
let	1	1
me	1	2
noble	2	1
so	1	1
the	2	1
the	2	1
told	1	1
you	2	1
was	2	1
was	2	1
with	1	1
you	2	1

III. SEARCH TIME COMPARISION BETWEEN STRUCTURED AND SEMI-STRUCTURED DATA

To search for a particular key word in the general document ,in the structured data and in semi-structured data is compared in this section.

TABLE 7

COMPRISION OF STRUCTURED AND SEMI-STRUCTURED DATA

	Structured	**SemiStructured**
Technology	Relational databases:Tables	XML
Transaction Management	Using concurrency techniques	Adapted from RDBMS
Robustness	Very Robust	-
Query	SQL	Xquery
VersionManagement	Table	Graphs
scalability	difficult	simple

The semi-structured data occupies more space than structured data. For search operation, the semi-structured data takes long time when compared than structured data. The processing of data is very slow in semi-structured data when compared to structured data

To search for a particular key word in the general document ,in the structured data and in semi-structured data is compared:

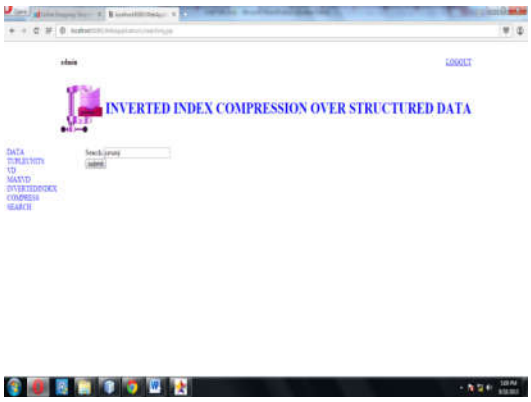

Fig .9 Giving a word to search

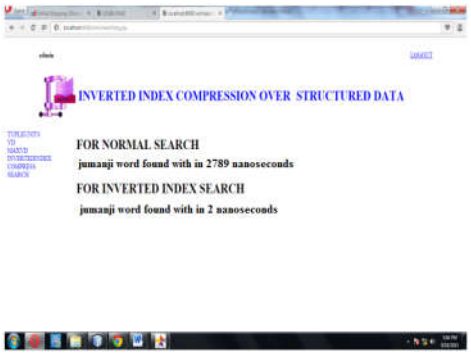

Fig.10 The word jumanji is found in structured data

Fig.11 The word jumanji is found in semi-structured data

For search operation, the semi-structured data takes long time when compared than structured data. The processing of data is very slow in semi-structured data when compared to structured data.

IV. RETRIEVING PAGES BASED ON RECALL AND PRECISION

For a Web search engine, compute precisions, returned pages for precisions for the top 5, 10, 15, 20, 25 and 30.

Table 8

Precision and Recall values for queries

Search field	precision	Recall
Doc1	0.700925926	0.9166667
Doc2	0.159775154	0.9666667
Doc3	1	0.875
Doc4	0.445478933	0.764238
average	0.5765455003	0.880643

Table 9

Precision and Recall values for documents

no	Doc:no#	Precision	Recall
1	588	1(1/1)	0.167
2	589	1(2/2)	0.333(2/6)
3	590	0.75(3/4)	0.5(3/6)
4	592	0.667(4/6)	0.667(4/6)
5	772	0.38(5/13)	0.833(5/6)

From the contingency matrix we can calculate the precision and recall

	Relevant	Not Relevant
Retrieved	TP(true positive)	FP(false positive)
Not Retrieved	FN(false negative)	TN(true negative)

Eg:

	Relevant	Not Relevant
Retrieved	TP(true positive)=10	FP(false positive)=30
Not Retrieved	FN(false negative)=5	TN(true negative)=55

Precision=0.25(10/10+30)

Recall=0.66(10/(10+5))

V. CONCLUSION

Inverted index was constructed both with structured data and semi-structured data. A clear comparison between the Structured and semi-structured data is given in this paper. The semi-structured data occupies more space than structured data. For search operation, the semi-structured data takes long time when compared than structured data. The processing of data is very slow in semi-structured data when compared to structured data. In the future, a plan to develop query processing techniques i.e. search in the compressed inverted index, and checking the compression times and decompression times, displaying all the synonyms during the query processing.

REFERENCES

[1]B.USHARANI, M.TANOOJKUMAR "SURVEY ON INVERTED INDEX COMPRESSION OVER STRUCTURED DATA" BY IJARCS, 4TH NATIONAL CONFERENCE ON RECENT TRENDS IN INFORMATION TECHNOLOGY 2015 P.NO 57-61.

[2]B.Usharani, M.TanoojKumar " Inverted Index Compression Over Structured Data" By IJCSE, National Conference on Advancements in Embedded Systems and Sensor Networks 2015 P.No 119-124.

[3]B.Usharani"InvertedIndexCompressionoverSemi-StructuredData" , National Conference on Internet of Things by AcharyaNagarjunaUniversity on March 3&4 2016.

[4]B.Usharani "Survey on Inverted Index Compression Over Semi-Structured Data" by IJIRC, ISSN: 2455-2275(E) P.No 97-104

[5]B.Usharani" Comparison between Structured and Semi-Structured data for Inverted Index Compression"by IJRCSE , Ist National Conference on Recent Trends in Advanced Computing" 2016 p.no7-11

[6]http://en.wikipedia.org/wiki/Search_engine_indexing#The_forward_index

[7]http://googleblog.blogspot.in/2008/07/we-knew-web-was-big.html

[8]"A low redundancy strategy for keyword search in structured and semi-structured data".J.L Lopez-venna,victor J.S,Ivan Lopez-Arevalo,Elsevier Aug 2014**.**

[9] Q. Su, J. Widom, Indexing relational database content offline for efficient keyword-based search, in: Proceedings of the 9th International Database

Engineering and Application Symposium (IDEAS), 25–27 July, Montreal, Canada, 2005, pp. 297–306.

[10] A. Markowetz, Y. Yang, and D. Papadias." Reachability Indexes for Relational Keyword Search" In ICDE, 2009.

[11] K. Q. Pu and X. Yu. "Keyword query cleaning" PVLDB, 1(1), 2008.

[12] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano." Efficient keyword search across heterogeneous relational database". In ICDE, 2007.

[13] F. Shao, L. Guo, and C. Botev" Efficient Keyword Search over Virtual XML Views" In VLDB, 2007.

[14] Q. Shao, P. Sun, and Y. Chen. WISE" a workflow information search engine" In ICDE, 2009.

[15] Q. Su and J. Widom. "Indexing relational database content offline for efficient keyword-based search" In IDEAS, 2005.

[16] C. Sun, C.-Y. Chan, and A. Goenka." Multiway SLCA-based keyword search in XML data". In WWW, 2007.

[17] "Databases and IR: Perspectives of a SQL "guy. NSF Information and Data Management PI Workshop, 2003.

[18] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. "Learning to create data-integrating queries". PVLDB, 1(1), 2008.

[19] Y. Tao and J. X. Yu. "Finding Frequent Co-occurring Terms in Relational Keyword Search". In EDBT, 2009.

[20] S. Tata and G. M. Lohman." SQAK: doing more with keywords" In SIGMOD, 2008.

[21] T. Tran, S. Rudolph, P. Cimiano, and H. Wang. "Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data" In ICDE, 2009.

[22] Q. H. Vu, B. C. Ooi, D. Papadias, and A. K. H. Tung" A graph method for keyword-based selection of the top-k database". In SIGMOD, 2008.

[23] S. Wang, Z. Peng, J. Zhang, L. Qin, S. Wang, J. X. Yu, and B. Ding. NUITS" A novel user interface for efficient keyword search over databases" In VLDB, 2006.

[24] P. Wu, Y. Sismanis, and B. Reinwald. "Towards keyword-driven analytical processing" In SIGMOD, 2007.

[25] Y. Xu and Y. Papakonstantinou."Efficient keyword search for smallest LCAs in XML databases" In SIGMOD, 2005.

[26] Y. Xu and Y. Papakonstantinou. "Efficient LCA based Keyword Search in XML Data". In EDBT, 2008.

[27] B. Yu, G. Li, K. R. Sollins, and A. K. H. Tung. "Effective keyword-based selection of relational databases" In SIGMOD, 2007.

[28] D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. "Keyword Search in Spatial Databases: Towards Searching by Document" In ICDE, 2009.

[29] http://www.webopedia.com/TERM/S/structured_da ta.html

[30] Retrieving and Materializing Tuple Units for Effective Keyword Search over Relational Databases (2008) by Guoliang Li , Jianhua Feng , Lizhu Zhou