INTERNATIONAL JOURNAL FOR DEVELOPMENT OF COMPUTER SCIENCE & TECHNOLOGY ISSN-2320-7884 (ONLINE)
VOLUME-1, ISSUE-IV (June-July) IS NOW AVAILABLE AT: www.ijdcst.com ISSN-2321-0257 (PRINT)

Query Processing for Personal Management Systems Based On Multidimensional Search

Sri Redya Jadav M.Tech (Ph.D)[1], C.H.Veena (PH.D)[2], Prabhakara Rao Thota[3]

#1 Professor, Bomma Institute of Technology and Science, Allipuram,Khammam (Dt),Andhra Pradesh.

#2 Asst.Professor, Bomma Institute of Technology and Science, Allipuram,Khammam (Dt),Andhra Pradesh.

#3 Student, Bomma Institute of Technology and Science, Allipuram,Khammam (Dt),Andhra Pradesh.

Abstract: Recently, parallel search engines have been implemented based on scalable distributed file systems such as Google File System. Existing tools typically support some IR-style ranking on the textual part of the query, but only consider structure (e.g., file directory) and metadata (e.g., date, file type) as filtering conditions. We propose a novel multi-dimensional search approach that allows users to perform fuzzy searches for structure and metadata conditions in addition to keyword conditions. Our techniques individually score each dimension and integrate the three dimension scores into a meaningful unified score. We also design indexes and algorithms to efficiently identify the most relevant files that match multi-dimensional queries. We perform a thorough experimental evaluation of our approach and show that our relaxation and scoring framework for fuzzy query conditions in noncontent dimensions can significantly improve ranking accuracy.

Index Terms: Multi-dimensional Search, Query Processing, Information seeking, search, orienteering, teleporting, context.

I. INTRODUCTION

Researchers have tried to support directed search by attempting to build a "perfect" search engine—i.e., one that returns exactly what is sought given a fully specified information need. Attempts to build such a search engine have focused on improving on keyword search by permitting users to better specify their information need through meta-data , natural language , and even context. Numerous search tools have been developed to perform keyword searches and locate personal information stored in file systems, such as the commercial tools Google Desktop Search and Spotlight. However, these tools usually support some form of *ranking* for the textual part of the query—similar to what has been done in the Information Retrieval (IR) community—but only consider structure (e.g., file directory) and metadata (e.g., date, file type) as *filtering* conditions. Although earlier studies of directed search focused on keyword search, most of the search behavior we observed did not involve keyword search. Instead of jumping directly to their information target using keywords, our participants navigated to their target with small, local steps using their contextual knowledge as a guide, even when they knew exactly what they were looking for in advance. This stepping behavior was especially common for participants with unstructured information organization. Searching for electronic information can be a complex, multistage process, where the information need evolves throughout the course of the search. However, often the search target is known in advance (e.g., a phone number or address). Such small, directed searches have been assumed to be simpler than large, evolving information seeking activities.

Recently, the research community has turned its focus on search over to Personal Information and Data spaces which consist of heterogeneous data collections. However, as is the case with commercial file system search tools, these works focus on IR-style keyword queries and use other system information only to guide the keyword-based search. We argue that allowing flexible conditions on structure and metadata can significantly increase the quality and usefulness of search results in many search scenarios.

We found that our participants used keyword search in only 39% of their searches, despite almost always knowing their information need up front. Instead of trying to jump directly to their information target using keyword search as might be expected, our participants performed directed situated navigation, similar to the Micronesian islanders' situated navigation described by Such man. To understand how to build the best possible search tool, we conducted an observational study of people performing personally motivated searches within their own information spaces.

In this case, by using the date, size, and structure conditions not as filtering conditions but as part of the ranking conditions of the query, we ensure that the best answers are returned as part of the search result. we propose a novel approach that allows users to efficiently perform fuzzy searches across three different dimensions: content, metadata, and structure. We describe individual *IDF*-based scoring approaches for each dimension and present a unified scoring framework for multi-dimensional queries over personal information file systems. We also present new data structures and index construction optimizations to make finding and scoring fuzzy matches efficient. While our work could be extended to a variety of data space applications and queries, we focus on a file search scenario in this paper. That is, we consider the granularity of the search results to be a single file in the personal information system. Of course, our techniques could be extended to a more flexible query model where pieces of data within files (such as individual. Finally, the user might misremember the directory path under which the file was stored.

II. RELATED WORK

For example, Marchionini detailed the importance of browsing in information seeking and O'Day and Jeffries characterized the information seeking process by outlining common "triggers" and "stop conditions" that guide people's search behaviors as their information needs change.

Information seeking—where a person's information need evolves throughout the search process—has been well studied. However, such studies introduce artificialities that can bias behavior. For example, the search tasks are imposed by the researcher rather than motivated by the user, and task has been shown to affect search performance. To gain a more realistic idea of what search is like in the real world, other studies have examined Web logs. Our study is unique in that we focus on directed search and look at behavior across a broad class of electronic types, including email, files, and the Web. By focusing on the communalities of interaction across types, we gain a broader understanding of general search techniques. Query log analysis provides insight into the types of information people search for (e.g., sex) and a cursory understanding of how people search (e.g., they use very short queries), but does not provide insight into their underlying intentions.

2.1 Tree Pattern Queries

Every query in the figure is also shown in the form of a tree pattern together with a Boolean formula imposing constraints on nodes in the tree.

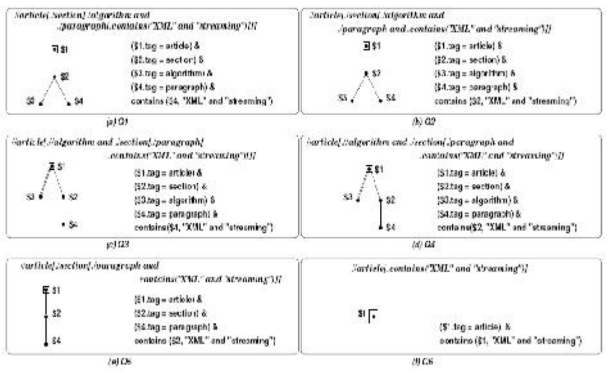
//article[./section[./algorithm and ./paragraph[.contains("XML" and "streaming")]]]
$1
$2
$3
$4
($1.tag = article) &
($2.tag = section) &
($3.tag = algorithm) &
($4.tag = paragraph) &
contains ($4, "XML" and "streaming")
(a) Q1
//article[./section[./algorithm and ./paragraph and .contains("XML" and "streaming")]]
($1.tag = article) &
($2.tag = section) &
($3.tag = algorithm) &
($4.tag = paragraph) &
contains ($2, "XML" and "streaming")
(b) Q2
//article[.//algorithm and ./section[./paragraph[.contains("XML" and "streaming")]]]
($1.tag = article) &
($2.tag = section) &
($3.tag = algorithm) &
($4.tag = paragraph) &
contains($4, "XML" and "streaming")
(c) Q3
//article[.//algorithm and ./section[.//paragraph and .contains("XML" and "streaming")]]
($1.tag = article) &
($2.tag = section) &
($3.tag = algorithm) &
($4.tag = paragraph) &
contains($2, "XML" and "streaming")
(d) Q4
//article[./section[.//paragraph and .contains("XML" and "streaming")]]
($1.tag = article) &
($2.tag = section) &
($4.tag = paragraph) &
contains ($2, "XML" and "streaming")
(e) Q5
//article[.contains("XML" and "streaming")]
($1.tag = article) &
contains ($1, "XML" and "streaming")
(f) Q6

Figure 1: Example Queries

Tree pattern queries constitute an important and expressive subset of XPath and make our illustration easier. Single edges denote parent child containment, while double edges denote ancestor-descendant containment.

These works are aimed at providing users with generic and flexible data models to accessing and storing information beyond what is supported in traditional files system. These differ from ours in that they attempt to leverage additional semantic information to locate relevant files while our focus is in determining the most relevant piece of information based solely on a user-provided query. Another study investigates user behavior when searching emails, files, and the web. Even if users know exactly what they are looking for, they often navigate to their target in small steps, using contextual information such as metadata information, instead of keyword-based search. Query log analysis provides insight into the types of information people search for (e.g., sex) and a cursory understanding of how people search (e.g., they use very short queries), but does not provide insight into their underlying intentions.

III. EXISTING APPROACH

Consider a user saving personal information in the file system of a personal computing device. In addition to the actual file content, structural location information (e.g., directory structure) and a potentially large amount of metadata information (e.g., access time, file type) are also stored by the file system.

In such a scenario, the user might want to ask the query:

*[filetype = *.doc AND*
createdDate = 03/21/2007 AND
content = "proposal draft" AND
structure = /docs/Wayfinder/proposals]

Let us description for above example the user might not remember the exact creation date of the file of interest but remembers that it was created *around* 03/21/2007. Similarly, the user might be primarily interested in files of type **.doc* but might also want to consider relevant files of different but related types (e.g., **.tex* or **.txt*).

The challenge is then to score answers by taking into account flexibility in the textual component *together* with flexibility in the structural and metadata components of the query. Once a good scoring mechanism is chosen, efficient algorithms to identify the best query results, *without considering all the data in the system*, are also needed.

IV. OUR PROPOSED APPROACH

We propose *IDF*-based scoring mechanisms for content, metadata, and structure, and a framework to combine individual dimension scores into a unified multi-dimensional score. We adapt existing top-*k* query processing algorithms and propose optimizations to improve access to the structure dimension index. Our optimizations take into account the top-*k* evaluation strategy to focus on building only the parts of the index that is most relevant to the query processing. We evaluate our scoring framework experimentally and show that our approach has the potential to significantly improve search accuracy over current filtering approached. We empirically demonstrate the effect of our optimizations on query processing time and show that our optimizations drastically improve query efficiency and result in good scalability.

4.1 QUERY PROCESSING

. It takes as input several sorted lists, each containing the system's objects (files in our scenario) sorted in descending order according to their relevance scores for a particular attribute (dimension in our scenario), and dynamically accesses the sorted lists until the threshold condition is met to find the *k* best answers. Random accesses occur when TA chooses a file from a particular list corresponding to some dimension, then needs the scores for the file in all the other dimensions to compute its unified score.

4.2 Evaluating Content Scores

Random accesses are supported via standard inverted list implementations, where, for each query term, we can easily look up the term frequency in the entire file system as well as in a particular file. We support sorted accesses by keeping the inverted lists in sorted order; that is, for the set of files that contain a particular term, we keep the files in sorted order according to their TF scores, normalized by file size, for that term.

4.3 Evaluating Metadata Scores

Each parent contains a larger range of values than its children, which ensures that the matches are returned in decreasing order of metadata scores. Similar to the content dimension, we use the TA algorithm recursively to return files in sorted order for queries that contain multiple metadata conditions.

Several techniques for XML query processing have focused on path matching. Most notably, the *Path Stack* algorithm iterates through possible matches using stacks, each corresponding to a query path component in a fixed order. To match a query path that allows permutations (because of node inversion) for some of its components, we need to consider all possible permutations of these components (and their corresponding stacks) and a directory match for a node group may start and end with any one of the node group components.

4.4 Improving Sorted Accesses

Evaluating queries with structure conditions using the lazy DAG building algorithm can lead to significant query evaluation times as it is common for multi-dimensional top*k* processing to access very relaxed structure matches

Algorithm 1 DAGJump(srcNode)

```
1. $s \Leftarrow$ getScore(srcNode)
2. currentNode $\Leftarrow$ srcNode
3. **loop**
4. targetDepth $\Leftarrow$ getDepth(currentNode)
5. childNode $\Leftarrow$ firstChild(currentNode)
6. **if** getScore(childNode) _= $s$ or hasNoChildNodes(childNode) **then**
7. exit loop
8. currentNode $\Leftarrow$ childNode
9. **for** each $n$ s.t. getDepth($n$) = targetDepth and getScore($n$) = $s$ **do**
10. Evaluate bottom-up from $n$ and identify ancestor node set
$S$ s.t. getScore($m$) = $s$, $\forall m \in S$
11. **for** each $m \in S$ **do**
12. **for** each $n\_$ on path $p \in$ getPaths($n,m$) **do**
13. setScore($n\_$, $s$)
14. setSkippable($n\_$)
15. **if** notSkippable($m$) **then**
16. setSkippable($m$)
```

We propose Algorithm 1, *DAGJump* It includes two steps: (a) starting at a node corresponding to a query path *P*, the algorithm performs a depth-first traversal and scoring of the DAG until it finds a parentchild pair, *P* " and *child*(*P*"), where *scoreidf* (*child*(*P*")) < *scoreidf* (*P*); and (b) score each node *P*" at the same depth (distance from the root) as *P”* ; if *scoreidf* (*P”*) = *scoreidf* (*P*).

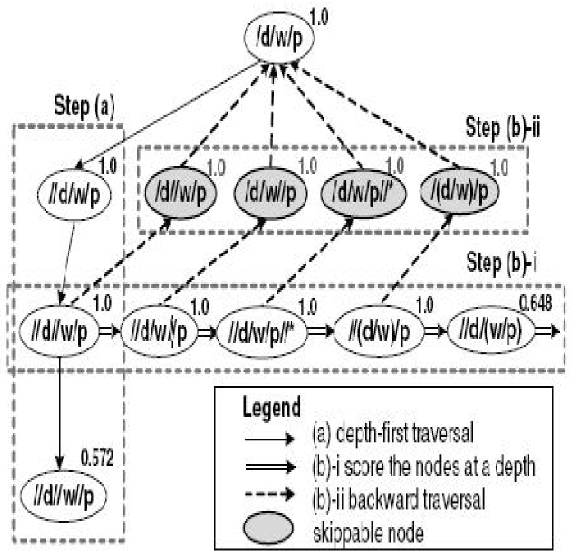

Figure 2: An example execution of *DAGJump*.

The two steps from Algorithm 1 are performed as follows: (a) starting at the root with a score of 1, *DAGJump* performs a depth-first traversal and scores the DAG nodes until it finds a node with a smaller score than 1 (*//d//w//p*); and (b) *DAGJump* traverses each node at the same depth as *//d//w/p* (the parent node of *//d//w//p*); for the four such nodes that have a score 1, *DAGJump* marks as skippable all nodes that are on their path to the root node.

Top-*k* query processing requires random accesses to the DAG. Using sorted access to emulate random access tends to be very inefficient as it is likely the top-*k* algorithm will access a file that is in a directory that only matches a very relaxed version of the structure condition, resulting in most of the DAG being materialized and scored.

V. PERFORMANCE EVALUATION

All experiments were performed using a prototype system implemented in Java. We use the MySql DB to persistently store all indexes and Lucene to rank content. Experiments were run on a PC with a 64-bit hyper-threaded 2.8 GHz Intel Xeon processor, 2 GB of memory, and a 10K RPM 70 GB SCSI disk, running the Linux 2.6.16 kernel and Sun’s Java 1.7 JVM.

5.1 Impact of Flexible Multi-Dimensional Search

We begin by exploring the potential of our approach to improve scoring (and so ranking) accuracy using two example search scenarios. In each scenario, we initially construct a content-only query intended to retrieve a specific target file and then expand this query along several other dimensions. In the first example, the target file is the novel “The Time Machine” by H. G. Wells, located in the directory path */Personal/Ebooks/Novels/*, and the set of query content terms in our initial content-only query Q1 contains the two terms *time* and *machine*. While the query is quite reasonable, the terms are generic enough that they appear in many files, leading to a ranking of 18 for the target file. Query Q2 augments Q1 with the exact matching values for file type, modification date, and containing directory. This brings the rank of the target file to 1. The remaining queries explore what happens when we provide an incorrect value for the non-content dimensions. potential impact of the node inversion relaxation. Specifically, queries Q23 and Q26 in the second example misorder the structure conditions as */Java/Mail* and */Java/Code*, respectively, compared to the real pathname */Personal/Mail/Code/Java*. Node inversion allow these conditions to be relaxed to *//(Java//Mail)* and *//(Java//Code)*, so that the target file is still ranked 1.

5.2 Comparing with Existing Search Tools

We compare the accuracy of our multi-dimensional approach with TopX [28], a related approach designed for XML search, Google Desktop Search (GDS), and Lucene.

Query sets: We consider a set of 40 synthetically generated search scenarios similar to those considered in the last section. Specifically, 20 emails and 20 XML documents (e.g., ebooks) were randomly chosen to be search targets. Choosing XML documents (emails are stored in XML format) allows internal structure to be included in TopX queries.

For our multi-dimensional approach, each query targeting a file *f* contains content, metadata, and structure conditions as follows:

• ***Content:*** 2 to 4 terms chosen randomly from *f*'s content.

• ***Metadata:*** A date (last modified) randomly chosen from a small range (±7 days to represent cases where users are searching for files they recently worked on) or a large range (±3 months to represent cases where users are searching for files that they have not worked on for a while and so only vaguely remember the last modified times) around *f*'s actual last modified date.

• ***Structure:*** a partial path *p* is formed by the correct ordering of 2 to 4 terms randomly chosen from *f*'s parent directory pathname.

5.3 Base Case Query Processing Performance

We now turn to evaluating the search performance of our system. We first report query processing times for the base case where the system constructs and evaluates a structural DAG sequentially without incorporating the DAGJump and Random DAG optimization algorithms.

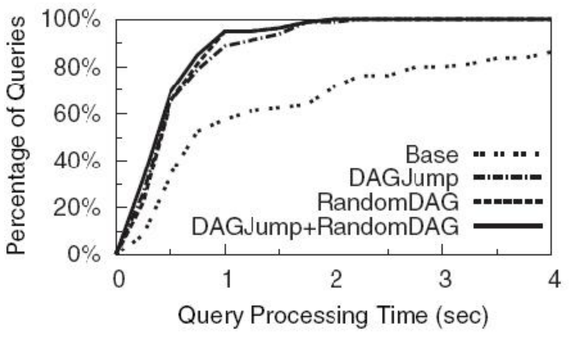

Figure 3: The CDFs of query processing time

Query set: We expand the query set used in the last section for this study. Specifically, we add targets and queries for 40 additional search scenarios, 20 targeting additional (mostly non-XML) documents and 20 targeting media files (music, etc.).

Choosing: *k***.** Query performance is a function of *k*, the number of top ranked results that should be returned to the user. We consider two factors in choosing a *k* value: (1) the mean recall (as defined above) and MRR, and (2) the likelihood that users would actually look through all *k* returned answers for the target file.

5.4 Query Processing Performance with Optimizations

We observe that these optimizations significantly reduce the query processing times for most of these queries. In particular, the query processing time of the slowest query, Q10, decreased from 125.57 to 1.96 seconds.

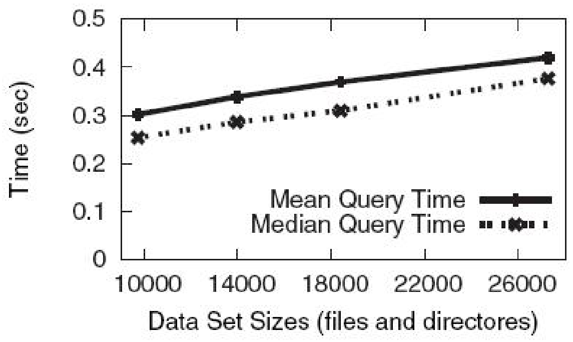

Figure 4: The mean and median query times for queries targeting email and documents plotted as a function of data set size.

To summarize, our DAGJump algorithm improves query performance when (a) there are many skippable nodes which otherwise would have to be scored during the top-*k* sorted accesses, and (b) the total processing time spent on these nodes is significant. The RandomDAG algorithm improves query performance when (a) the top*k* evaluation requests many random access, and (b) the total processing time that would have been spent on nodes successfully skipped by RandomDAG is significant.

VI. CONCLUSION

We defined structure and metadata relaxations and proposed *IDF*-based scoring approaches for content, metadata, and structure query conditions. This uniformity of scoring allows individual dimension scores to be easily aggregated. We have also designed indexing structures and dynamic index construction and query processing optimizations to support efficient evaluation of multi-dimensional queries. Our evaluation show that our multi-dimensional score aggregation technique preserves the properties of individual dimension scores and has the potential to significantly improve ranking accuracy. We also show that our indexes and optimizations are necessary to make multi-dimensional searches efficient enough for practical everyday usage.

VII. REFERENCES

[1] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML Documents via XML Fragments. In *Proc. of the ACM Intl.*

Conference on Research and Development in Information Retrieval (SIGIR), 2003.

[2] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping? In *Proc. Of the Conference on Innovative Data Systems Research (CIDR)*, 2005.

[3] J. Chen, H. Guo, W. Wu, and C. Xie. Search Your Memory! – An Associative Memory Based Desktop Search System. In *Proc. of the ACM Intl. Conference on Management of Data (SIGMOD)*, 2009.

[4] J.-P. Dittrich and M. A. Vaz Salles. iDM: A Unified and Versatile Data Model for Personal Data space Management. In *Proc. of the Intl. Conference on Very Large Databases (VLDB)*, 2006.

[5] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 2003.

[6] M. Franklin, A. Halevy, and D. Maier. From Databases to Data spaces: a New Abstraction for Information Management. *SIGMOD Record*, 34(4), 2005.

About Authors:

Sri Redya Jadav M.Tech, (Ph.D) Head of the Department Department of CSE & IT . BE from Osmania University , M.Tech from JNTU University and pursuing PhD from JNTU university. Having 12 years of experience in teaching Computer Technologies published various research articles in National and International Journals.

CH.Veena,B.Tech,M.tech,(PH.D)
Assistant Professor,CSE
Research scholar in Data Mining.

The author named **Prabhakara rao thota** pursuing his **M.Tech** from Bomma College, Khammam affiliated to **JNTUH.** His research interests are **Network security, data warehousing, and operating systems.**