Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 10, October 2013, pg.91 – 97

RESEARCH ARTICLE

Unified Multidimensional Fuzzy Search Using Optimizing Query Processing

V. Mahesh[1], U. Vijay Kumar[2]

[1]M. Tech, Dept of CSE, ASCET, Gudur, India

[2]Associate Professor, Dept of CSE, ASCET, Gudur, India

[1] mahesh.v540@gmail.com; [2] vjay39@gmail.com

Abstract— In this paper the large amount of semi structured data users access and store in personal information, there is a critical need for commercial search tools using to retrieve different types of data in a simple way. Existing tools are support IR-style ranking on the textual part of the query, but only consider structure and metadata as filtering conditions. We proposed tools Google Desktop Search tools are typically support a multidimensional fuzzy search approach that allows users to perform searches for structure and metadata conditions in addition to keyword conditions. And our techniques independently score about above three dimensions and integrate the scores into a meaningful united score. The scoring mechanism depends on the Inverse Document Frequency (IDF). We adapt existing top-k query processing algorithms and optimization to improve access structure dimension. We evaluate our scoring frame work experimentally and significantly improve search accuracy. In this we show that our query processing strategies perform and scale well, making our fuzzy search approach practical for every day usage.

Keywords: - Information retrieval; multidimensional search; Query processing; IDF; Google Desktop search

I. INTRODUCTION

The large amount of data access and stored in personal information management systems is rapidly increasing the growth in capacity and dropping price of storage. This type of information is driving a critical need for commercial search tools to access different types of data in a simple and efficient manner. Commercial tools are providing both high-quality scoring mechanisms and efficient query processing capabilities. Various search tools are using to search keywords and locate personal information stored in file systems, such tools are commercial tools as Google Desktop Search (GDS) and Spotlight [1]. 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 and meta data as filtering conditions. Recently, the research community has turned its focus on search over to Personal Information and Data spaces [10] 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.

Keyword-only searches are often insufficient, as illustrated by the following example:

Example 1. 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: createdDate ¼ 03=21=2007 AND content ¼ 00proposal draft00 AND structure ¼ =docs=Wayfinder=proposals Current tools would answer this query by returning all files of type _:doc created on 03/21/2007 under thedirectory /docs/Wayfinder/proposals (filtering conditions) that have content similar to "proposal draft" (ranking expression), ranked based on how close the content matches "proposal draft" using some underlying text scoring mechanism. Because all information other than content are used only as filtering conditions, files that are very relevant to the query, but which do not satisfy these exact conditions would be ignored. For example, documents created on 03/19/2007 and files in the directory / archive/proposals/Way finder containing the terms "proposal draft" would not be returned. We argue that allowing flexible conditions on structure and metadata can significantly increase the quality and usefulness of search results in many search scenarios. For instance, in Example 1, 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). Finally, the user might misremember the directory path under which the file was stored.

II. RELATED WORK

Several works have focused on the user perspective of personal information management [10]. These works allow users to organize personal data semantically by creating associations between files or data entities and then leveraging these associations to enhance search. Other works address information management by proposing generic data models for heterogeneous and evolving information. 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. Instead, we focus on querying information that is already present in the file system. An interesting future direction would be to include entity associations in our search and scoring framework. Other file system related projects have tried to enhance the quality of search within file system by leveraging the context in which information is accessed to find related Information or by altering the model of the file system to a more object-orientated database system [8].

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.

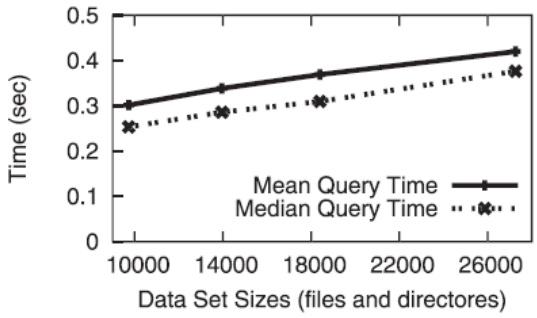


Fig. The mean and median query times for queries targeting e-mail and documents plotted as a function of data set size

Recently there has been a surge in projects attempting to improve desktop search. These projects provide search capabilities over content and but use other query dimensions, such as size, date, or types, as filtering conditions. Research has shown that desktop users tend to prefer location-based search to a keyword-based search which observes that users rely on the information such as directories, dates, and names when searching files. Another study investigates user behavior when searching e-mails, 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. These studies motivate the need for personal information management system search tools that use metadata and structure information. The Initiative for the Evaluation of XML retrieval (INEX) promotes new scoring methods and retrieval techniques for XML data[11]. INEX provides a collection of documents as a testbed for various scoring methods in the same spirit as TREC was designed for keyword queries. While many methods have been proposed in INEX, they focus on content retrieval and typically use XML structure as a filtering condition. As a result, the INEX data set s and queries would need to be extended to account for structural heterogeneity. Therefore, they could not be used to validate our scoring methods. As part of the INEX effort, XIRQL presents a content-based XML retrieval query language based on a probabilistic approach. While XIRQL allows for some structural vagueness, it only

considers edge generalization, as well as some semantic generalizations of the XML elements. Similarly, JuruXML provides a simple approximate structure matching by allowing users to specify path expressions along with query keywords and modifies vector space scoring by incorporating a similarity measure based on the difference in length, referred to as length normalization.

III. MULTIDIMENSIONAL SCORING

In this section, we present our unified framework for assigning scores to files based on how closely they match query conditions within different query dimensions. We distinguish three scoring dimensions: content for conditions on the textual content of the files, metadata for conditions on the system information related to the files, and structure for conditions on the directory path to access the file. We represent files and their associated metadata and structure information as XML documents. We use a simplified version of XQuery to express metadata and structure conditions in addition to keyword-based content conditions. Any file that is relevant to one or more of the query conditions is then a potential answer for the query; it gets assigned a score for each dimension based on how close it matches the corresponding query condition. Scores across multiple dimensions are unified into a single overall score for ranking of answers. Our scoring strategy is based on an IDF-based interpretation of scores, as described below. For each query condition, we score files based on the least relaxed form of the condition that each file matches. Scoring along all dimensions is uniformly IDF-based which permits us to meaningfully aggregate multiple single-dimensional scores into a unified multidimensional score.

3.1 Scoring Content

We use standard IR relaxation and scoring techniques for content query conditions. Specifically, we adopt the scoring formulas from Lucene, a state-of-the-art keyword search tool. These formulas are as follows: where Q is the content query condition, f is the file being scored, N is the total number of files, Nt is the number of files containing the term t, and NormLengthðfÞ is a normalizing factor that is a function of f's length.2 Note that relaxation is an integral part of the above formulas since they score all files that contain a subset of the terms in the query condition.

3.2 Scoring Metadata

We introduce a hierarchical relaxation approach for each type of searchable metadata to support scoring. For example, Fig. 1 shows (a portion of) the relaxation levels for file types, represented as a DAG.3 Each leaf represents a specific file

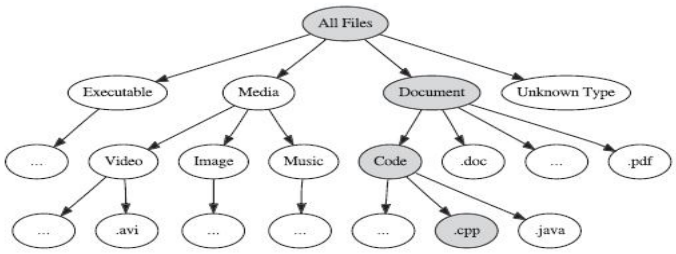


Fig. 1. Fragment of the relaxation DAG for file type (extension) metadata.

3.3 Scoring Structure

Most users use a hierarchical directory structure to organize their files. When searching for a particular file, a user may often remember some of the components of the containing directory path and their approximate ordering rather than the exact path itself. Thus, allowing for some approximation on structure query conditions is desirable because it allows users to leverage their partial memory to help the search engine locate the desired file.

3.4 Score Aggregation

We aggregate the above single-dimensional scores into a unified multidimensional score to provide a unified ranking of files relevant to a multidimensional query. To do this, we construct a query vector, ~ VQ, having a value of 1 (exact match) for each dimension and a file vector, ~ VF , consisting of the single-dimensional scores of file F with respect to query Q. (Scores for the content dimension is normalized against the highest score for that query condition to get values in the range ½0; 1_.) We then compute the projection of ~ VF onto ~ VQ and the length of the resulting vector is used as the aggregated score of file F. In its current form, this is simply a linear combination of the component scores with equal weighting. The vector projection method, however, provides a framework for future exploration of more complex aggregations.

IV. OPTIMIZED QUERY PROCESSING

We adapt an existing algorithm called the Threshold Algorithm (TA) to drive query processing. TA uses a threshold condition to avoid evaluating all possible matches to a query, focusing instead on identifying the k best answers. 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.

Critically, TA relies on sorted and random accesses to retrieve individual attribute scores. Sorted accesses, that is, accesses to the sorted lists mentioned above, require the files to be returned in descending order of their scores for a particular dimension. Random accesses require the computation of a score for a particular dimension for any given file.

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. To use TA in our scenario, our indexing structures and algorithms need to support both sorted and random access for each of the three dimensions. We now present these indexing structures and algorithms.

4.1 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.4Wethen use the TA algorithm recursively to return files in sorted order according to their content scores for queries that contain more than one term.

4.2 Evaluating Metadata Scores

Sorted access for a metadata condition is implemented using the appropriate relaxation DAG index. First, exact matches are identified by identifying the deepest DAG node N that matches the given metadata condition (see Section 2.2). Once all exact matches have been retrieved from N's leaf descendants, approximate matches are produced by traversing up the DAG to consider more approximate matches. 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. Random accesses for a metadata condition require locating in the appropriate DAG index the closest common ancestor of the deepest node that matches the condition and the deepest node that matches the file's metadata attribute (see Section 2.2). This is implemented as an efficient DAG traversal algorithm.

4.3 Evaluating Structure Scores

The structure score of a file for a query condition depends on how close the directory in which the file is stored matches the condition. All files within the same directory will therefore have the same structure score. To compute the structure score of a file f in a directory d that matches the (exact or relaxed) structure condition P of a given query, we first have to identify all the directory paths, including d, that match P. For the rest of the section, we will use structure condition to refer to the original condition of a particular query and query path to refer to a possibly relaxed form of the original condition. We then sum the number of files contained in all the directories matching P to compute the structure score of these files for the query using (6). The score computation step itself is straightforward; the complexity resides in the directory matching step. Node inversions complicate matching query paths with directories, as all possible permutations have to be considered. Specific techniques and their supporting index structures need to be developed. Several techniques for XML query processing have focused on path matching. Most notably, the Path Stack algorithm [9] iterates through possible matches using stacks, each corresponding to a query path component in a fixed order.

V. QUERY PROCESSING IN THE STRUCTURE DIMENSION

In this section, we present our dynamic indexes and algorithms for efficient processing of query conditions in the structure dimension. This dimension brings the following challenges:

The DAGs representing relaxations of structure conditions [4] are query dependent and so have to be built at query processing time. However, since these DAGs grow exponentially with query size, i.e., the number of components in the query, efficient index building and traversal techniques are critical issues. . The TA algorithm requires efficient sorted and random access to the single-dimension scores (Section 3). In particular, random accesses can be very expensive. We need efficient indexes and traversal algorithms that support both types of access. We propose the following techniques and algorithms to address the above challenges. We incrementally build the query dependent DAG structures at query time, only materializing those DAG nodes necessary to

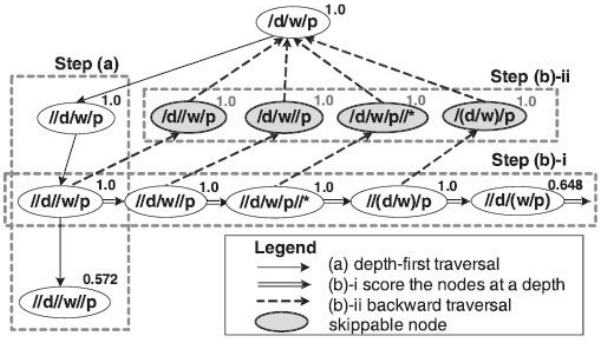
answer a query (Section 4.1). To improve sorted access efficiency, we propose techniques to skip the scoring of unneeded DAG nodes by taking advantage of the containment property of the DAG (Section 4.2). We improve random accesses using a novel algorithm that efficiently locates and evaluates only the parts of the DAG that match the file requested by each random access

5.1 Incremental Identification of Relaxed Matches

As mentioned in Section 2.3, we represent all possible relaxations of a query condition and corresponding IDF scores using a DAG structure. Scoring an entire query relaxation DAG can be expensive as they grow exponentially with the size of the query condition. For example, there are 5, 21, 94, 427, and 1,946 nodes in the respective complete DAGs for query conditions =a; =a=b, =a=b=c; =a=b=c=d; =a=b=c=d=e. However, in many cases, enough query matches will be found near the top of the DAG, and a large portion of the DAG will not need to be scored. Thus, we use a lazy evaluation approach to incrementally build the DAG, expanding and scoring DAG nodes to produce additional matches when needed in a greedy fashion. The partial evaluation should nevertheless ensure that directories (and therefore files) are returned in the order of their scores. For a simple top-k evaluation on the structure condition, our lazy DAG building algorithm is applied and stops when k matches are identified. For complex queries involving multiple dimensions, the algorithm can be used for sorted access to the structure condition. Random accesses are more problematic as they may access any node in the DAG. The DAG building algorithm can be used for random access, but any random access may lead to the materialization and scoring of a large part of the DAG.5 In the next sections, we will discuss techniques to optimize sorted and random access to the query relaxation DAG.

5.2 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 multidimensional top k processing to access very relaxed structure matches, i.e., matches to relaxed query paths that lay at the bottom of the DAG, to compute the top-k answers.



VI. EXPERIMENTAL RESULTS

We now experimentally evaluate the potential for our multidimensional fuzzy search approach to improve relevance ranking. We also report on search performance achievable using our indexing structures, scoring algorithms, and top-k adaptation.

6.1 Experimental Setup

Experimental environment. All experiments were performed using a prototype system implemented in Java. We use the Berkeley DB to persistently store all indexes and Lucene to rank content. Experiments were run on a PC with a 64-bit hyperthreaded 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 1.4.2 JVM. Reported query processing times are averages of 40 runs, after 40 warm-up runs to avoid measurement JIT effects. All caches (except for any Berkeley DB internal caches) are flushed at the beginning of each run. Data set. As noted in there is a lack of synthetic data sets and benchmarks to evaluate search over personal information management systems. Thus, we use a data set that contains files and directories from the working environment of one of the authors. This data set contains 14.3 GB of data from 24,926 files organized in 2,338 directories; 24 percent of the files are multimedia files (e.g., music and pictures), 17 percent document files (e.g., pdf, text, and MS Office), 14 percent e-mail messages,7and 12 percent source code files. The average directory depth was 3.4 with the longest being 9. On average, each directory contains 11.6 subdirectories and files, with the largest—a folder containing e-mails—containing 1,013. The system extracted 347,448 unique stemmed content terms.8 File modification dates span

10 years. Seventy five percent of the files are smaller than 177 KB, and 95 percent of the files are smaller than 4.1 MB.

6.2 Impact of Flexible Multidimensional 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. For each query, we consider the ranking of the target file by our approach together with whether the target file would be ranked at all by today's typical filtering approaches on non-content query conditions. A representative sample of results is given in Table 1. 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. For example, in query Q10, a couple of correct but wrongly ordered components in the directory name still bring the ranking up to 1. In contrast, if such directories were given as filtering conditions, the target file would be considered irrelevant to the query and not ranked at all; queries which contain a "_" next to our technique's rank result represent those in which the target file would not be considered as a relevant answer given today's typical filtering approach.

Results for the second example, which is a search for an e-mail, are similar.

Query Evaluation Results						
	Query Conditions					Comments on Relaxation
Query	Content	Type	Modification Date	Structure	Rank	from Query Q1/Q14
Search Scenario 1: The user searches for the electronic book "The Time Machine". **Target file:** /Personal/Ebooks/Novels/The Time Machine.pdf **Structure condition:** /Personal/Ebooks/Novels (abbreviated as /p/e/n, 'c' in the incorrect path stands for 'Comics')						
Q1	time, machine	-	-	-	18	Base Query
Q2	time, machine	.pdf	22 Jan 07 18:09	$/p/e/n$	1	Correct Values (all dimensions)
Q3	time, machine	.pdf	-	-	9	Correct File Type
Q4	time, machine	.doc	-	-	42 *	Incorrect File Type
Q5	time, machine	Docs.	-	-	18	Relaxed Range
Q6	time, machine	-	21-27 Jan 07	-	1	Relaxed Range (Week of month)
Q7	time, machine	-	23 Jan 07 18:09	-	1 *	Incorrect Date (off by 1 day)
Q8	time, machine	-	15 Feb 07 18:09	-	4 *	Incorrect Date (off by 1 month)
Q9	time, machine	-	-	$/p/e/n$	1	Correct Path
Q10	time, machine	-	-	$/n/e$	1 *	Incorrect Order/Correct Components
Q11	time, machine	-	-	$/p/e/c$	2 *	Incorrect Path
Q12	time, machine	Docs.	Jan 07	$/p/e$	1	Relaxed Range (all dimensions)
Q13	time, machine	.pdf	15 Feb 07 18:09	$/c/e$	1 *	Incorrect Date and Path
Search Scenario 2: The user searches for an email that discusses the java implementation of the IR algorithm for the file system Wayfinder. **Target file:** /Personal/Mail/Code/Java/920-SA_____-20061018192157-78.xml **Structure condition:** /Mail/Code/Java (abbreviated as /m/c/j, 'p' in the incorrect path stands for 'Python')						
Q14	Wayfinder, IR	-	-	-	42	Base Query
Q15	Wayfinder, IR	.xml	18 Oct 06 14:21	$/m/c/j$	1	Correct Values (all dimensions)
Q16	Wayfinder, IR	.xml	-	-	35	Correct File Type
Q17	Wayfinder, IR	.txt	-	-	39 *	Incorrect File Type
Q18	Wayfinder, IR	Docs.	-	-	39	Relaxed Range
Q19	Wayfinder, IR	-	15-21 Oct 06	-	1	Relaxed Range (Week of month)
Q20	Wayfinder, IR	-	17 Oct 06 14:21	-	1 *	Incorrect Date (off by 1 day)
Q21	Wayfinder, IR	-	18 Nov 06 14:21	-	8 *	Incorrect Date (off by 1 month)
Q22	Wayfinder, IR	-	-	$/m/c/j$	1	Correct Path
Q23	Wayfinder, IR	-	-	$/j/m$	1 *	Incorrect Order/Correct Components
Q24	Wayfinder, IR	-	-	$/m/c/p$	1 *	Incorrect Path
Q25	Wayfinder, IR	Docs.	Oct 06	$/m/c$	1	Relaxed Range (all dimensions)
Q26	Wayfinder, IR	.txt	18 Nov 06 14:21	$/j/c$	1 *	Incorrect Date and Path

Table1.The Rank of target Files Returned by a set of Related Queries

VII. CONCLUSION AND FUTURE WORK

We presented a scoring framework for multidimensional queries over personal information management systems. Specifically, 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 multidimensional queries.

We implemented and evaluated our scoring framework and query processing techniques. Our evaluation show that our multidimensional 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 multidimensional searches efficient enough for practical everyday usage. Our optimized query processing strategies exhibit good behavior across all dimensions, resulting in good overall query performance and scalability.

REFERENCES

[1] Wei Wang, Christopher Peery, Amelie Marian, and Thu D. Nguyen " Efficient Multidimensional Fuzzy search for Personal Information management System" vol. 24 SEPTEMBER 2012.
[2] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis, "Automated Ranking of Database Query Results," Proc. First Biennial Conf. Innovative Data Systems Research (CIDR '03), 2003.
[3] S. Amer-Yahia, P. Case, T. Ro¨ lleke, J. Shanmugasundaram, and G. Weikum, "Report on the DB/Ir Panel at SIGMOD 2005," SIGMOD Record, vol. 34, no. 4, pp. 71-74, 2005.
[4] S. Amer-Yahia, S. Cho, and D. Srivastava., "Tree Pattern Relaxation," Proc. Int'l Conf. Extending Database Technology (EDBT), 2002.
[5] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman, "Structure and Content Scoring for XML," Proc. Int'l Conf. Very Large Databases (VLDB), 2005.
[6] S. Amer-Yahia, L.V.S. Lakshmanan, and S. Pandit, "FleXPath: Flexible Structure and Full-Text Querying for XML," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD), 2004.
[7] Lucene, http://lucene.apache.org, 2012.
[8] R.A. Baeza-Yates and M.P. Consens, "The Continued Saga of DB-IR Integration," Proc. Int'l Conf. Very Large Databases (VLDB), 2004.
[9] C.M. Bowman, C. Dharap, M. Baruah, B. Camargo, and S. Potti, "A File System for Information Management," Proc. Int'l Conf. Intelligent Information Management Systems (ISMM), 1994.
[10] N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal Xml Pattern Matching," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD), 2002. 1596 IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 9, SEPTEMBER 2012
[11] Y. Cai, X.L. Dong, A. Halevy, J.M. Liu, and J. Madhavan, "Personal Information Management with SEMEX," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD), 2005.

