SPARK: A Keyword Search Engine on Relational Databases

Yi Luo, Wei Wang, Xuemin Lin

School of Computer Science and Engineering
University of New South Wales
Australia

{luoyi, weiw, lxue}@cse.unsw.edu.au

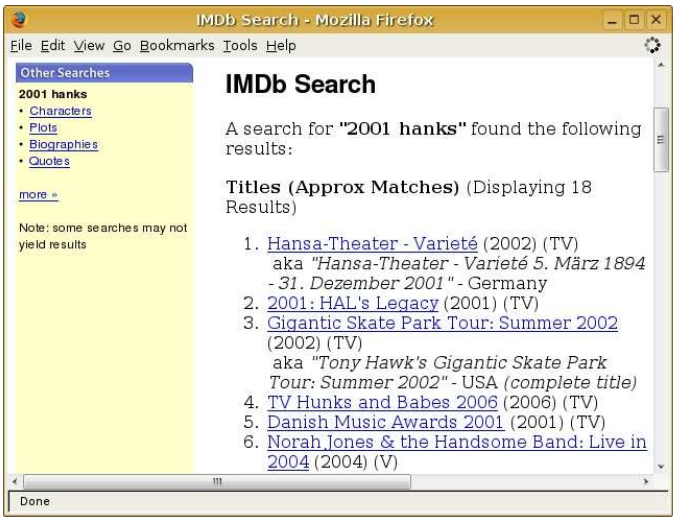
I. Introduction

Relational database is the most widely adopted and mature technology for information storage. As many services on the Web (e.g., blog and wiki sites) and advanced applications (e.g., Customer Relationship Management Systems and Content Management Systems) are built on RDBMSs, increasing amount of text data is now stored in relational databases, accompanied by increasing demands of retrieving relevant information by free-style keyword search.

Current commercial solutions to keyword search for relational databases are *not* sufficient. A typical solution is to build a set of query templates that map keyword search to full-text matching within one or more attributes. For example, `imdb.com` allows users to search for matching movies according to title, actor, etc. This solution offers only limited query capabilities and flexibilities. For instance, we performed a search of "`2001 hanks`" using the search interface on `imdb.com` and cannot find any relevant answer (See Figure 1(a)).

SPARK is a system that we developed recently to address this issue. It is able to assemble matching tuples from *different* relations together according to foreign key to primary key relationships such that the tuples are *collectively* relevant to the query. For example, we show the top-3 results for the same query returned by the SPARK system populated with `imdb.com`'s data. The second and the third results are obvious relevant to the query, but won't be found unless we join **Movies**, **ActorPlay**, and **Actor** together. The technical challenge lies in how to find a general-purpose and effective ranking method for the search results while optimizing the search within a potentially huge search space. In our previous work [1], we approached this problem by proposing a novel ranking method that takes into consideration several important ranking factors. We also demonstrated that, with two new algorithms, our query processing speed could be up to two orders of magnitude faster than alternative methods. These results are the foundation of the SPARK system.

In this demo, we will demonstrate the SPARK system in work. In particular, we demonstrate that it can help both *casual users* and *professional users* to locate relevant information via keyword search in databases. This goal is achieved by (a) offering an easy-to-use search interface and intuitive result


IMDb Search - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Other Searches
2001 hanks
Characters
Plots
Biographies
Quotes
more »
Done

Note: some searches may not yield results

IMDb Search

A search for **"2001 hanks"** found the following results:

Titles (Approx Matches) (Displaying 18 Results)

1. Hansa-Theater - Varieté (2002) (TV)
 aka *"Hansa-Theater - Varieté 5. März 1894 - 31. Dezember 2001"* - Germany
2. 2001: HAL's Legacy (2001) (TV)
3. Gigantic Skate Park Tour: Summer 2002 (2002) (TV)
 aka *"Tony Hawk's Gigantic Skate Park Tour: Summer 2002"* - USA *(complete title)*
4. TV Hunks and Babes 2006 (2006) (TV)
5. Danish Music Awards 2001 (2001) (TV)
6. Norah Jones & the Handsome Band: Live in 2004 (2004) (V)

(a) Results from `imdb.com`

1	**Movies**: "Primetime Glick" (2001) Tom Hanks/Ben Stiller (#2.1)
2	**Movies**: "Primetime Glick" (2001) Tom Hanks/Ben Stiller (#2.1) $\leftarrow$ **ActorPlay**: Character = Himself $\rightarrow$ **Actors**: Hanks, Tom
3	**Actors**: John Hanks $\leftarrow$ **ActorPlay**: Character = Alexander Kerst $\rightarrow$ **Movies**: Rosamunde Pilcher - Wind über dem Fluss (2001)

(b) Top-3 Results on SPARK (Relation Names are in Bold Font and Arrows denote Foreign Key to Primary Key Relationships)

Fig. 1. Searching `2001 hanks`

ranking for casual users to access information stored in a relational database, and (b) providing *advanced* search methods for professional users to perform data searching and analysis.

This proposal differs from other demos on the same topic and our previous work [1] in the following aspects:

- We emphasize on the search effectiveness of the system. Result quality is one of the most important factors for keyword search systems, yet it has *not* been given sufficient attentions in the past. We will demonstrate several ranking functions implemented in SPARK. Users can appreciate the relative strength and weakness of different ranking methods on several datasets in the demo.

- We emphasize on how some new search and browsing methods (in addition to the generic method in our previous work [1]) can better serve user's information need. In particular, we will demonstrate how a *new* OLAP-style result browing method can help users to locate the desired information quickly — a typical search scenario we identified in the user study (See Section III-C.4). New advanced search features and their implementation details will also be introduced in the demo (See Sections III-C and III-D).
- We demostrate the performance of the system powered by different query processing algorithms. Users can have an intuitive feeling of the substantial differences in query response time by different algorithms and the trade-offs of search quality/specifications and the search speed.

II. System Architecture

We plot the architecture of the SPARK system in Figure 2. We choose a server-browser architecture. This means no specialized software needs to be installed and users only need a browser to access the service. SPARK Server is implemented in Java and uses adapters based on JDBC to communicate with relational DBMSs. Using adapters hides the difference in the capabilities of the underlying RDBMSs and allows the server to be *plugged* into any supported RDBMSs easily. Currently, we support adapters to Oracle and MySQL.

The SPARK Server contains three key components: a non-free tuple set constructor, a candidate network generator, and a query processor.

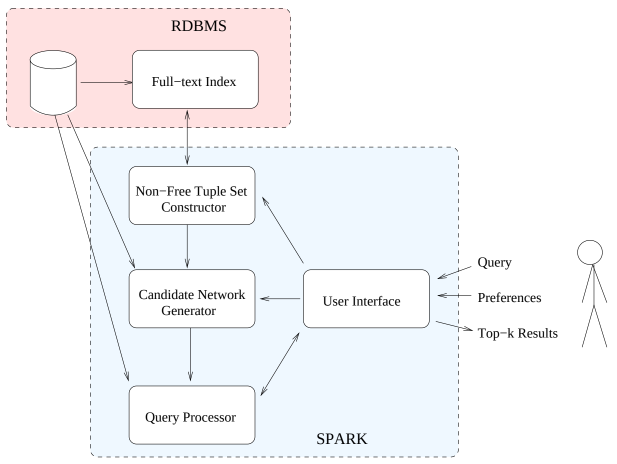


Fig. 2. System Architecture

a) Non-Free Tuple Set Constructor: A *non-free tuple set* of relation R is all the tuples in R that contain at least a match to a query keyword.

When a user inputs a query, each keyword in the query is immediately sent here to generate the corresponding non-free tuple sets in all relations. Since full-text index is built on all relations, a non-free tuple set can be constructed by merging the postings of the inverted index.

b) CN Generator: A *candidate network* (CN) [2] is a relational algebra expression over the tuple sets, such that the corresponding query may produce some query results. A tuple set is either a non-free tuple set or a free tuple set (i.e., all tuples in a relation).

The CN Generator computes a set of CNs by a breadth-first subgraph enumeration algorithm adapted from [2]. It finds all *minimal* CNs whose sizes are within a user-defined threshold. A CN is minimal if all its leaves correspond to non-free tuple sets, i.e., contain some keywords.

c) Query Processor: We implemented four algorithms in the query processor. Sparse and Global Pipeline are due to [3]; Skyline Sweeping and Block Pipeline are proposed in our previous work [1].

Basically, Sparse processes each CN by sending a single SQL query to RDBMS. It iteratively chooses a CN and sorts its results, until top-k results are found. Global Pipeline pushes the top-k constraints deep into the query execution. Skyline Sweeping and Block Pipeline are designed to minimize unnecessary join checking to a further extent.

Compared to our previous work [1], we have also significantly extended the Block Pipeline algorithm by utilizing materialized results and sharing of computations among CNs.

III. Demo Description

In the demonstration, we will (a) motivate the keyword search problem in the relational database context, and demonstrate several novel use of the system to satisfy a variety of information demands; (b) introduce several SPARK's features and showcase their usage in helping user locating the desired information; and (c) demonstrate a set of experiments that evaluate performance of the system, and compare results of different implementations. We give further details below.

A. Introduction and Datasets

The first part of the demo will be an overview of the keyword search problem in the relational database context. We will compare results returned by the same keyword query from `imdb.com`, `google.com`, and SPARK. We will also illustrate that subtle and unexpected relationships can be found by keyword search.

We will then introduce the datasets used in the demo.

DBLP It is a bibliographic database comprosed of 6 tables and around 0.9 million tuples in total.

IMDB We download the data from `imdb.com`'s web site, and convert a subset of the data into relational tables, including: *movies*, *direct*, *directors*, *actressplay*, *actresses*, *actorplay*, *actors*, and *genres*. There are about 10 million tuples in total.

Mondial The database contains geographical information from various resources. There are 28 tables and around 17,000 tuples.

Northwind Northwind is a small sample database that comes with Microsoft Access. It contains sales data for a fictitious food trading company, with tables for customers, orders, suppliers, products, etc. It contains 8 tables and around 3,200 tuples.

These datasets are chosen to cover a large spectrum of applications, data size, complexity of the schema, etc.

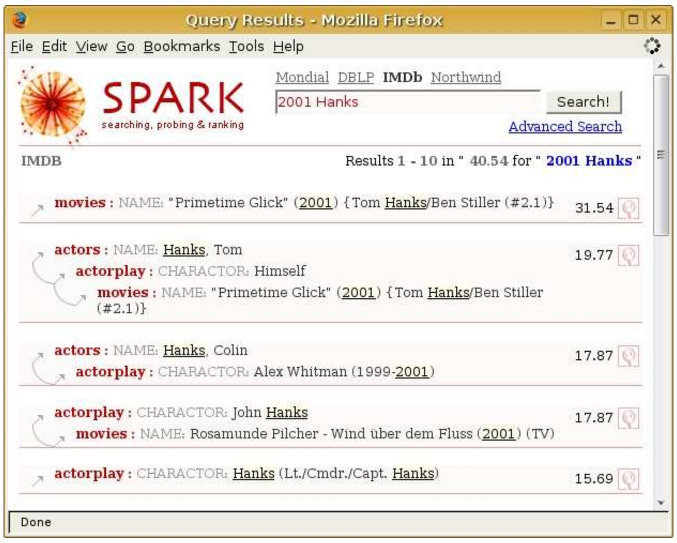
SPARK supports two search modes: *simple search* and *power search*. They are elaborated in the following two subsections.

B. Simple Search

The simple search is intended for casual users. The design principle is to be simplistic and intuitive.

At SPARK's homepage, a user can (a) select a data source (among Mondial, DBLP, IMDB, and Northwind), and (b) input a keyword query. The default output is a ranked list of search results assembled from tuples in the select data source that are relevant to the query. Results are scored using the ranking functions in [1].

For example, Figure 3 shows the results of the query `2001 hanks` in decreasing order of their relevance scores (which is displayed to the right of the search results)


Query Results - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
SPARK
searching, probing & ranking
Mondial DBLP IMDb Northwind
2001 Hanks
Search!
Advanced Search
Done

IMDB Results 1 - 10 in " 40.54 for " 2001 Hanks "

movies : NAME: "Primetime Glick" (2001) {Tom Hanks/Ben Stiller (#2.1)} 31.54

actors : NAME: Hanks, Tom 19.77
actorplay : CHARACTOR: Himself
movies : NAME: "Primetime Glick" (2001) {Tom Hanks/Ben Stiller (#2.1)}

actors : NAME: Hanks, Colin 17.87
actorplay : CHARACTOR: Alex Whitman (1999-2001)

actorplay : CHARACTOR: John Hanks 17.87
movies : NAME: Rosamunde Pilcher - Wind über dem Fluss (2001) (TV)

actorplay : CHARACTOR: Hanks (Lt./Cmdr./Capt. Hanks) 15.69

Fig. 3. Ranked Results of Query `2001 hanks`

C. Power Search

The power search is intended for professional users. The design goal is to be customizable and to provide advanced features to search for information more easily.

1) Query Specification: In addition to simple keyword queries, SPARK supports several advanced query specifications:

Conjunction and Negation By default, SPARK assumes the OR semantics. User can override this by specifying a `+` before a keyword. If a keyword is prefixed with "`-`", we consider the user do not want any result containing the keyword.

Phrase Search and Wildcard Matching Keywords appearing within a pair of quotes (`' '`) will be treated as a phrase, which must be matched together in a tuple in a search result. Wildcards (`*` and `.`) are also supported.

Schema Term and Alias In order to fine-tune the occurrence of a keyword match, a keyword can be prefixed with a table name, an attribute name, their combination, or an alias defined by system administrators. E.g., query `actor.name:cage` will only match actor names containing `cage`, but not any actress or director; query `person-name:cage` will match all person's names, if the alias `person-name` is defined to include `actor.name`, `actress.name`, and `director.name`. In the special case where only schema term appears, it denotes that the result must contain the particular schema object.

The rich set of advanced query specification can help professional users to express their information need more specifically by injecting their domain knowledge into the query. It is also helpful for keywords that occur frequently or are ambiguous.

2) Query Evaluation Algorithm: Users can also fine-tune the query evaluation algorithm.

Maximum CN Size This parameter sets the threshold on the largest candidate network that can be searched by the system.

Query Evaluation Algorithm User can choose from Block Pipeline (default), Skyline Sweeping, Global Pipeline, and Sparse algorithms.

Top-k User can choose a k value such that the query evaluation algorithm is optimized to return top-k results fast.

3) Ranking: The demo uses the novel ranking formula proposed in our previous work [1]. The new ranking method assigns a score to each query result by considering three factors:

IR Score We adapt the state-of-the-art IR ranking formula to query results by modelling query results as *virtual documents*, and all the results obtained by the same join expression without full-text selection conditions as a *virtual document collection*. The new method fixes the common problem in previous approaches that overly reward the contribution of the same keyword appearing in different components in the result.

Completeness We use a completeness factor quantifying the degree of matching for a result to a query, based on the *extended Boolean model*. Results matching part of the keywords will get a lower completeness factor value. A tunable parameter p is used to allow the ranking switching between OR and AND semantics. A smaller value (e.g., $p = 1.0$) simulates the OR semantics, while a larger value, such as $p = 2.0$, approximates the AND semantics well in our experiments.

Size We adopt an alternative result size normalization factor. The factor takes into consideration (a) the semantical role each tuple set plays in the result, and (b) the distribution of result sizes for the given query. Therefore, the size normalization factor is adaptive to the database schema and the query.

We have shown in [1] that our new ranking method outperforms previous approaches substantially. The average *reciprocal rank*[1] of SPARK is often close to the maximum value 1.0, while previous approaches are always below 0.5.

In the demo, we will show the impact of parameter settings on the search results. For example, for the query `partition patel david` on the DBLP dataset, the result of "*David J. DeWitt and Jignesh M. Patel coauthored a paper named Partition Based Spatial-Merge Join*" is ranked as the 7th under the default ranking parameter $p = 1.0$. If we increase p to 1.4 and then 2.0, this result moves to the second and the first, respectively.

4) OLAP-style Result Browsing: Two novel ranking and browsing methods have been implemented to enable OLAP-style exploration of query results.

Group by CNs Under this mode, we assume results from different CNs are incomparable. Thus, top-k results are computed for each CN. Only the top-1 result of each CN, together with a "snippet" about the statistics of the other matches, is shown as the search result. Once the user clicks a CN, all the top-k results from the chosen CN will be shown.

Expand the Search Users can select and mark part of a search result, and choose to expand the result. The semantics is to return a superset of selected tuples that is relevant to the query. This provides an easy way to quickly zoom into part of the database and locate relevant information.

We use the following example to illustrate the OLAP-style Result Browsing. Assume a user wants to find all papers written by `David Dewitt` on `join`s, but cannot remember his last name. She issues a query `david join`, which has many search results in the DBLP dataset, as both keywords are very popular. She can first group results according to their CNs, and only "drill down" to CNs of interest to her, i.e., $Author \bowtie Writes \bowtie Paper$. The top-3 results are three `join`-related papers from `david dewitt`, `david scot taylor`, and `david vineyard`, respectively. Seeing that `david dewitt` is the author she wants, she can *expand* the search by fixing the author tuple whose name is `david dewitt`. Consequently, all the `join`-related paper from `david dewitt`, even though many of them are not ranked as top-k results globally, can be found.

D. Demonstrating System Performances

We will demonstrate a set of experiments that are designed to evaluate performance of the system. They evaluate both the effectiveness and efficiency. We will compare with two previous work [3], [4]. We have command line interface supporting parameters to run the search using the ranking method and query evaluation method in [3], [4] and SPARK. Search results from different ranking algorithms can be compared side by side.

[1]It is calculated as the inverse of the rank of the *first* relevant results.

The experiments will showcase a number of features of the SPARK system, including:

- SPARK can return high quality result for most queries even in the default mode (i.e., using only the simple search interface).
- Thanks to the Block Pipeline algorithm, the query evaluation speed of SPARK is usually an order of magnitude faster than the better of SPARSE and Global Pipeline algorithm.
- Block pipeline algorithm running in progressive mode incurs minimal amount of waiting time of users, as the desired result is usually returned fast as the top-1 result.

REFERENCES

[1] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in *SIGMOD*, 2007.

[2] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword search in relational databases." in *VLDB*, 2002, pp. 670–681.

[3] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search over Relational Databases," in *VLDB*, 2003.

[4] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases." in *SIGMOD*, 2006, pp. 563–574.