Towards a Concurrent Implementation of Keyword Search Over Relational Databases

M.Sc. Thesis

Richard Drake

richard.drake@uoit.ca

Supervisor: Dr. Ken Q. Pu

University of Ontario Institute of Technology
Oshawa, Ontario, Canada

18 June 2014

Motivation

- Two important data models:
  - The relational model is rigid in structure and highly normalized
  - The document model is flexible and provides keyword search
- Choice between data integrity and accessibility
- Why can't we have both?

Thesis Statement

A system could be built that is capable of transforming data from the relational model to the document model. The transformation is reversible, allowing the original data model to be recovered. This system would use the keyword search capabilities, *along with the relational information, to quickly discover related fragments of information.*

Research Goals

- Define a formal framework for transforming data from the relational to document model
- Design a collection of expressive query operators for analyzing text from relational data sets
- Perform graph search over the document model
- Investigate implementation techniques to make the query operators performant on modern, multicore machines

Relational Model

- A database is a collection of relations
- A relation is a set of named tuples
- Each named tuple consists of a set of attributes corresponding to values
- An entity group is a set of related tuples (joined by foreign key constraints)

Document Model

- A document collection is a set of documents
- A document may have one or more fields
- A field is a bag of tokens
- A prescribed lexical analyzer converts bodies of text into a bag of tokens

Framework Overview

- Indexing
  1. Iterate over all named tuples in relational database
  2. Convert each named tuple into a document
  3. Encode relational information into one or more indexing document(s)
- Search
  1. Fuzzy value search at the tuple-level
  2. Graph search for connections among entity groups

Iterate

- Each entity and entity group defined in a configuration file
- Configuration specifies the SQL query to retrieve all entities from database
- This query is executed and each row is converted

```
(crawl
  [this db-conn idx-w]
  (let [sql (S :sql)]
    (execute-query db-conn sql (fn [row] ...))))
```

Figure: Code to iterate over every named tuple in entity group

Convert

- Each attribute of a named tuple directly maps to a field in a document
- Analysis is performed on each attribute before storing in a field
- The framework supports multiple analyzers

$$\textsc{Attr}[t] \xrightarrow{\textit{analyzed}} \textsc{Field}[d] \tag{1}$$

$$\alpha_1, \alpha_2, \ldots, \alpha_n \xrightarrow{\textit{analyzed}} f_1, f_2, \ldots, f_n \tag{2}$$

Where α_i is the value of an attribute, and f_i are the resulting tokens in a document.

Encode

- Relational information between related named tuples is encoded in the indexing document
- This document, x, is the concatenation of unique identifiers of every related document

$$x[\text{"entities"}] = \{\mathrm{UID}[t] : t \in V(T)\} \tag{3}$$

Where T is the entity group, and $V(T)$ is the set of named tuples in the entity group.

Fuzzy Search at Tuple-Level

The encoding allows us to perform the following:

- Fuzzy search of relational attributes
- Search of named tuples in relations
- Find entity groups

Graph Search Over Document Space

- Use the indexing document to discover adjacent nodes
- Issue search query to discover all entity groups containing UID[u]

```
(boolean-query [[(query :type :group) :and]
                [(query :entities id) :and]])
```

Graph Search Algorithm and Implementation

- Chose Breadth-First Search (BFS) as search algorithm
- Adapted BFS to run concurrently without locking using Clojure's Software Transactional Memory (STM)
- Utilized Clojure's concurrency primitives (atoms, references) in the concurrent implementation of BFS

Data Corpus

- Derived from MyCampus data[1]

Relation	Attributes
Course	code, title, subject
Section	id, actual, campus, capacity, credits, levels, registration_start, registration_end, semester, sec_code, sec_number, year, course
Schedule	id, date_start, date_end, day, schedtype, hour_start, hour_end, min_start, min_end, classtype, location, section_id
Instructor	id, name
Teaches	id, schedule_id, instructor_id, position

Table: Subset of mycampus dataset schema

[1] `http://uoit.ca/mycampus/`

Benchmarking Queries

- Keywords are sampled from the database to form search queries
  - May not occur within the same entity group
  - Sampled from instructor name, course code, section id, etc.
- Search for connections between these keywords
  - Must utilize graph search to discover connections between keywords
  - Monitor distance (hops) between keywords and measure performance

Results

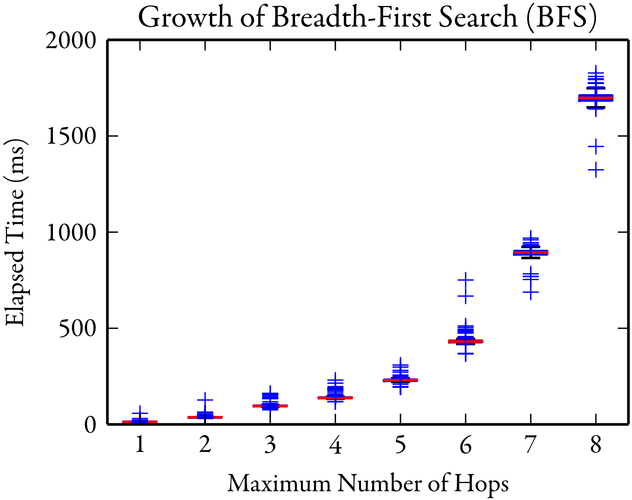

Figure: Growth of runtime of BFS, by hops

Results

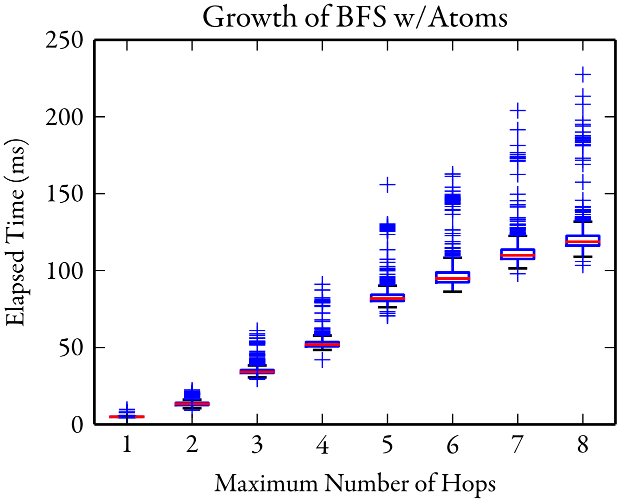

Figure: Growth of runtime of BFS with atoms, by hops

Results

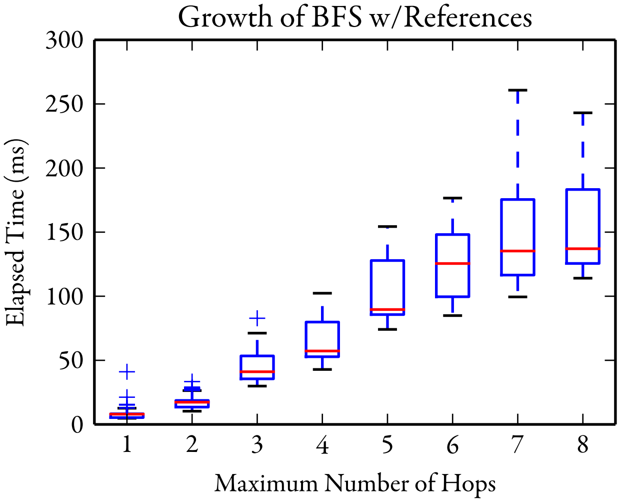

Figure: Growth of runtime of BFS with references, by hops

Results

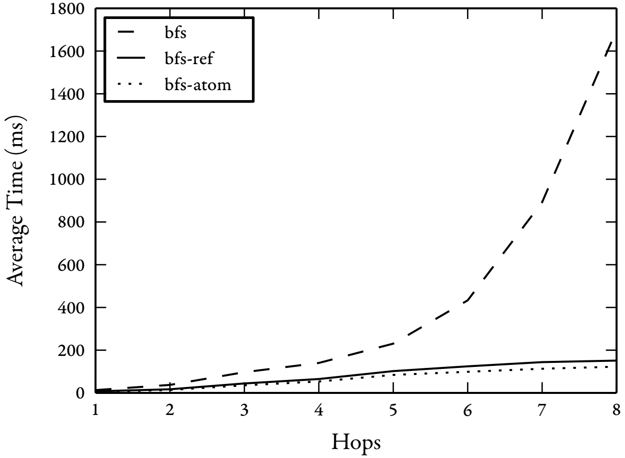

Figure: Growth of runtime of each implementation, by hops

Contributions

- Provided a framework to transform data from the relational to document model
- Demonstrated the reversibility of this transformation
- Utilized the flexibility of the document model
  - e.g. spelling correction, entity group search
- Performed graph search over document space
- Investigated the reduction in runtime from a concurrent graph search

Lessons Learned

- Simple algorithms are easiest to parallelize
- Clojure's STM implementation is simple and effective
- Clojure is powerful and encouraged correct code that was easier to optimize later

Publication

This work has been submitted to the 15th IEEE conference on Information Reuse and Integration.

Using Document Space For Relational Search

Richard Drake
Faculty of Science
University of Ontario Inst of Technology

Ken Q. Pu
Faculty of Science
University of Ontario Inst of Technology

***Abstract*—In this paper, we present a family of methods and algorithms to efficiently integrate text indexing and keyword search from information retrieval to support search in relational databases. We propose a bi-directional transformation that maps relational database instances to document collections. The transformation is shown to be a homomorphism of keyword search. Thus, any search of tuple networks by a keyword query can be efficiently executed as a search for documents, and vice versa. By this construction, we demonstrate that indexing and search technologies developed for documents can naturally be reduced and integrated into relational database systems.**

I. Motivation

Information retrieval has been an active and fruitful field of research since 1960's. With seminal work by [1] and [2], the IR community has laid the foundation of automatic text indexing and keyword query processing of text documents. The technology for document indexing continues to gain momentum with the growing presence of text data found on the Web and in social media. For instance, new techniques by [3] and [4] improve on the traditional similarity measures by incorporating further (NLP) on the context of phrases and words.

In the last decade, there has been a tremendous interest from the database community to support keyword search queries for structured relational databases. Systems such as *Discover* [5], *DBxplorer* [6] and *BANKS* [7] and many others [8], [9] model relational tuples as documents, and foreign key joins as links. Thus, it's possible to derive IR-style scoring function for document similarity. More recently, semantic information [10], schema and meta data [11] have been incorporated into the search algorithm.

$$\begin{tikzcd} \mathrm{DB} \arrow[r, "\textsc{Encode}"] \arrow[d, "\textsc{search}_{\mathrm{DB}}"'] & \mathrm{DOC} \arrow[d, "\textsc{search}_{\mathrm{DOC}}"] \\ \mathrm{DB} & \mathrm{DOC} \arrow[l, "\textsc{Decode}"] \end{tikzcd}$$

where $\textsc{search}_{\mathrm{DB}}$ and $\textsc{search}_{\mathrm{DOC}}$ are the search functions for relational databases and documents respectively.

Practical experiences have demonstrated that the state-of-the-art $\textsc{search}_{\mathrm{DOC}}$ has more performant implementations ([12], [13]) compared to its relational database counter part. Our interest is to construct efficient transformations h and g such that relational search $\textsc{search}_{\mathrm{DB}}$ can be efficiently implemented as a composition of $g \circ \textsc{search}_{\mathrm{DOC}} \circ h$, effectively taking advantage of the document search technology.

II. Search problems for relational and document models

In this section, we present the formal definition of relational databases and collections of documents. We also formalize the notion of keyword search entity graphs and join networks of entity graphs in relational databases.

A. Relational entities

A relational database consists of a collection of tables which are interconnected via linkages.

A table, T, has a number of attributes:

$$\mathrm{attr}(T)$$

A tuple in a table $r \in T$ is defined as a mapping from attributes to values:

Future Work

- Simplify configuration
- Incremental indexing
- Generalize results by benchmarking standard datasets

Towards a Concurrent Implementation of Keyword Search Over Relational Databases

M.Sc. Thesis

Richard Drake

richard.drake@uoit.ca

Supervisor: Dr. Ken Q. Pu

University of Ontario Institute of Technology
Oshawa, Ontario, Canada

18 June 2014