Molly

by

Richard Drake

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Masters of Science

in

Faculty of Science

Computer Science

University of Ontario Institute of Technology

Supervisor: Dr. Ken Q. Pu

September 2012

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Preface

Background and Motivation

The introduction of keyword search has revolutionized how we find information. Over the years, numerous techniques have been developed which make searching through large amounts of information for one or more keywords extremely fast. With the advent of faster computer hardware, we are able to not only search through small attributes of a document (eg. Title, synopsis, etc.) but rather the entirety of the document itself.

An example of an early system which utilized keyword search would be a library catalogue. Such a system would allow a user to search, by keyword, for the title of a book, manuscript, etc. The results would show item titles matching the keyword(s), as well as other information such as whether or not the item is in circulation, as well as where it is located within the library. This information would come from a relational database.

With the rise of the World Wide Web, much information was placed online. This information would be easy to access if one knew how to locate it. Unfortunately, over time, so much information existed on the World Wide Web that it became difficult to keep track of it all. There was a need to index all of this information and make it accessible. This need was filled by a Web search engine.

Initial Web search engines comprised of simple scripts that gathered listings of files on FTP servers; they were essentially link farms. A few short years later, the first full-text (keyword) search engine, WebCrawler, was released.

While full-text search engines provided an excellent means for locating information, as the Web grew larger, the volume of noise also grew larger. In addition, every Web page could be structured in a different way; the Web was largely a collection of unstructured documents. That is, there were few obvious links between them.

Search engines such as Google attempted to solve this problem by introducing new algorithms, such as PageRank, to rank Web pages on both the relevance of their content as well as their reputation. The idea was if a page is linked to often, it is considered to be more authoritative on a subject than a page with fewer links. This allowed the relevance of a page to be computed based

on not only its contents, but its artificial importance.

Molly attempts to avoid some of the issues plaguing search engines. It deals primarily with structured, filtered data. This allows us to provide results with less noise. In addition, the fact that it deals with structured data means links between documents are explicitly stated. Rather than inferring a link between documents based on hyperlinks, we know when two documents are linked together.

This thesis provides an overview of the Molly system.

Outline

Chapter 1 provides a look at systems which tried to solve a similar problem. It analyzes where other systems have failed and where they have succeeded.

Chapter 2 discusses the system design and rationale behind the design.

Chapter 3 details the implementation of Molly. It also discusses several factors which influence performance.

Chapter 4 moves beyond single-threaded performance and discusses making the system use multiple threads simultaneously. It also discusses what performance gains, if any, were made.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 0

Data Representation and Notation

Chapter 1

Literature Review

Chapter 2

System Design, Rationale, & Performance

2.1 Introduction

2.1.1 Data Representation

The data from the database is represented in various data structures. There are separate representations for each type of data: values, entities, and entity groups.

Value

Definition 1. A **Value** represents a single piece of information. To avoid repetition, each value is unique. That is, $\exists! \, v \in V$, where v is a value in the set V of all values.

Entity

Definition 2. An **Entity** is a collection of attributes, a_n, each mapped to a single value, v_n. An entity also includes additional information such as a unique identifier.

id	$T_n \vert v_{id}$
a_1	v_1
a_2	v_2
$\vdots$	$\vdots$
a_n	v_n

Figure 2.1: The structure of an entity

Entities are analogous to rows in a database table. Thus, the unique identifier is generated based on the table name, T_n, as well as unique key in the table, v_{id}. The unique key identifies the row, and the table name identifies the table. Together they uniquely identify the entity within the entire database.

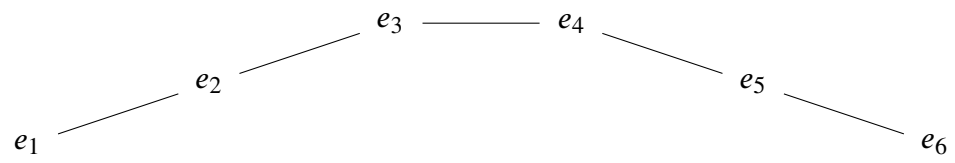
$\exists!\, e_{id} \in E$, where E is the set of all entities.

Entity Group

Definition 3. An **Entity Group** joins together two or more entities. These entity groups can also have attributes, a_n, and values, v_n, associated with them much like entities.

e_L	$[e_1, e_2, \ldots, e_n]$
a_1	v_1
a_2	v_2
$\vdots$	$\vdots$
a_n	v_n

Figure 2.2: The structure of an entity group



2.2 Overview of System Components

2.2.1 Crawler

2.2.2 Indexer

2.2.3 Core (Processing)

2.2.4 API

2.2.5 Frontend

2.3 Data Flow

2.4 Implementation Issues

2.5 Ford-Fulkerson

Ensure: $1 = 1$

Chapter 3

Blah blah blah

3.1 Choosing a Breadth-First Search Algorithm

We took several criteria into consideration when choosing a Breadth-First Search algorithm to perform the graph search. In the context of this problem, there is no obvious heuristic to help predict the distance between two nodes. We decided to use a constant cost function. That is, $d_{(v_1,v_2)} = n$. The selected algorithm must also be highly parallelizable.

BFS - Wrong result if non-uniform distance Bellman-Ford - Always correct result Dijkstra - Faster than Bellman-Ford, uses priority queue Push-Relabel & Johnson - Uses complicated (and difficult to parallelize) data structures A-Star Search - No obvious heuristic