

Combining Systems and Databases: A Search Engine Retrospective

Paper by: Eric Brewer
Presented by: Kristen LeFevre

1

Databases, Search Engines, and Systems, Oh My!

- Search engines (SEs) are systems for managing very large amounts of data
- Make little use of DBMS systems as we typically think of them
 - Relational data model; supports SQL, ACID, etc.
- Core Hypothesis: In retrospect, SEs should have been built using *principles* of databases, but not existing artifacts
 - Same may be true for other data-intensive systems

2

Key Database “Principles”

- **Top-Down Design**
 - Systems typically built “bottom-up” to deliver capabilities to unknown applications
 - E.g., OS, file systems, etc.
 - DBMSs built “top-down” starting with desired functionality (e.g., SQL, ACID semantics)
 - SEs are “whole” systems like DBMSs
- **Data Independence**
 - Data exists in sets, without pre-specified storage format or access methods
- **Declarative Query Language**
 - Queries specify “what” not “how”
 - Useful in SEs, even though queries are not SQL

3

Why Not Use a DBMS?

- Existing DBMSs built with different goals
 - Transactions, ACID semantics
 - SEs prioritize high-availability for queries; consistency less important
 - General-purpose query language (e.g., SQL) and query optimizer
 - SEs support of simpler, more limited language
- Result: Search engine implementation atop Informix (a relational DBMS) was very slow!

4

Crawl, Index, Serve

- Search engine uses a static (“snapshot”) database to serve read-only queries
 - Other work done offline
- Main Steps:
 1. Crawl: Visit and download pages (“documents”)
 2. Index: Parse and interpret a set of documents offline; move as much work as possible away from the online service step
 - Often includes scoring!
 3. Serve: Execute queries against static database

5

Query Language

- Basic query is a set of terms
 - $Q = \{w_1, w_2, \dots, w_k\}$
- Also boolean conditions
 - E.g., bay area -oakland lang:english
- Return a set of documents, ranked by score
 - Scoring mechanism isn’t the focus of this paper
 - Sample Score Function: $\text{Score}(Q,d) = \text{Quality}(d) + \sum_i \text{Score}(w_i, d)$
 - Quality measures something about the document (e.g., length, reviews, etc.)

6

Schema

Document table, D , about 3B rows

Docid	URL	Date	Size	Abstract
-------	-----	------	------	----------

Word table, about 1T rows:

WordID	Docid	Score	Position info
--------	-------	-------	---------------

Property table, about 100B rows:

WordID	Docid
--------	-------

Term table, T , about 10M rows:

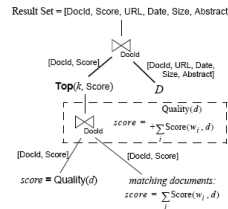
String	WordID	Stats
--------	--------	-------

- In retrospect, this is an appropriate simple schema

7

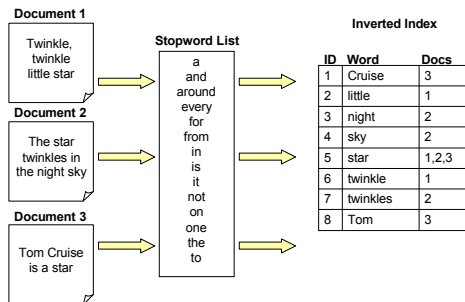
Query Processing

- One *logical* query plan
 - Access method: scan an *inverted index*
 - Sorted list of documents that contain each word
 - Cache intermediate results; no pipelining
 - Only need to use merge-join (on pre-sorted lists)
 - Query optimizer tries to exploit cached results, minimize the number of joins



8

Inverted Index



9

Parallelizing the Query

- General approach:
 - Data is (horizontally) partitioned across nodes
 - A subset of documents on each node
 - Run the same query on each node (on different data)
 - Master* node coordinates the plan for a particular query, sends sub-queries to *followers*, collates the result

10

Updates Overview

- Typical Update Process:
 - A "chunk" (a set of documents) is the atomic unit of update
 - Re-crawl, re-index, and refresh a chunk at a time
 - Main challenge is invalidating cached data
- Updates to single records (documents) are the exception
 - E.g., illegal content
 - Heavyweight operation

11

Fault Tolerance Overview

- Goal: High availability of the server (query processor)
 - Crawling and indexing is independent; needs not be highly available
- Can replicate some documents for availability, load balancing
 - Should replicate popular sites
 - Faults on non-replicated data simply reduce database size temporarily
 - Re-crawl later...
 - Very different from OLTP!!

12

Fault Tolerance Overview

- Disk Faults: New copies of failed data loaded in the background
- Master Faults: Restart the query with a different master
- Follower Faults: Timeout, restart on a replica if available
- Additional Challenges:
 - *Graceful degradation* if system is overwhelmed
 - Admission control, dynamically dropping data chunks
 - *Disaster recovery*

13

Conclusions (I)

- In retrospect, database principles are a convenient way of understanding search engines
- Why did Informix do so badly, compared to hand-optimized system?
 - ~10X slower in 1996
 - Key Problem: *Over-generalization*
 - Search engine didn't need: Locking, General-purpose query optimizer, Multiple join algorithms, Access control, Integrity constraints, ...

14

Conclusions (II)

- Brewer speculates that this will be a trend:
Data-intensive systems that utilize the "principles" of database, but customized for specific applications
- Similar observations from Stonebraker for data streams and sensor data
 - See "One size fits all: An idea whose time has come and gone"
- What do you think?

15