Combining Systems and databases: A Search Engine Retrospective.

In this paper the author describes how data-intensive systems like search engines that do not use a DBMS can be designed using the principles of databases.

Search engines (SEs) are arguably the largest data management systems in the world; although there are larger databases in total storage there is nothing close in query volume. A modern search engine handles billions of documents, terabytes of data, and handles upwards of 150 million queries per day. Search engines are constantly being modified to handle more data and give better results. Search is also becoming important in intranets and on local computers. So a paper that describes how to design a search engine using principles of database systems is very useful.

Problem Statement
Given:
• Requirement for a search engine
Find
• A search engine design including
  o A schema to store data
  o A query language
  o Implementation of query mechanism
Objectives
• Make use of database principles
• Efficient (Fast)
• Cost effective
• Scalable
Constraints
• Highly available

The author presents how Search Engines should be designed. He focuses on how Search Engines and other data intensive systems can use the principles of database systems without actually using database implementations. The paper describes why a DBMS can’t be used to store the data and search. Because of the large amount of data and large number of users it would be very inefficient, very costly and not highly available. The reason these performance gaps are due to the fact that DBMS are geared towards OLTP applications and not for text search. The DBMSs use locking, generalized query plans and limited caching. For a custom built search engine no locking is needed, aggressive caching can be used and query plan can be optimized for text search. As for
cost most DBMS are sold on a per user basis or per processor based. In both cases cost would be very high when used in a clustered, internet environment.

**Major Contributions**
The main contributions of the paper are:

- What principles of DBMS should be applied when designing data intensive applications like search engines.
- Reasons why SEs can’t be implemented using DBMS
  - Speed: DBMS are slow
  - Cost: DBMS are not cost effective, given the amount of data and number of users.
  - High Availability: SEs prefer high availability over consistency.

- Overview of SE design:
  - Crawl, Index, Serve
  - A schema for storing data
  - Queries (read only)
    - Scoring of documents
    - Making a query plan
    - Query implementation
      - Access methods and physical operators
      - Optimization using caching & multiway joins
      - Implementation on cluster with replication and range partitioning of data
      - Other optimizations like compression of indices and pre-loading of cache.
    - Data update mechanism using chunks
    - Real time deletion using ‘deleted’ property
  - Fault tolerance
    - Disk faults – dealt by reloading
  - SE challenges not faced by traditional DBMS
    - Personalization – using cookies or a row in database for each user
    - Logging – logging large amount of data and querying it
    - Query Rewriting – Based on language, personal data or source document
    - Phrase queries – proximity affects scoring of documents

The most important contribution is describing how search engines have requirements different than traditional databases and why traditional implementations can’t be used. However the principles of database systems can be applied to search engines.

**Key Concepts**
Search engines use a crawler the web and retrieve documents. Then an indexer parses and interprets collection of documents. Its output is a portion of static database called a chunk, that reflects scoring and normalization for those documents. The server simply executes the queries against a collection of chunks. It doesn’t have to revisit the documents to score them again.
Documents contain words and have properties. Words have a score and properties are boolean. Examples of properties include ‘primary language’, whether it contains images etc. A query is a set of terms, which could be either a word or property.

The score of a document \( d \) for query \( Q \) is defined as

\[
\text{Score}(Q,d) = \text{Quality}(d) + \sum_{i} \text{Score}(w_i, d)
\]

The quality of document is independent of query words and reflects things like length (shorter is generally better), popularity of document or site, incoming links, quality of the containing site, and external reviews. The score for each word is a determined at index time and depends on frequency and location (such as in the title or headings or metadata, or bold).

Given this simplified scoring plan the paper describes how to devise a query plan for a given query. The schema shown in Figure 1 is used.

<table>
<thead>
<tr>
<th>Document table, ( D ), about 3B rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocId</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word table, about 11 rows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordId</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property table, about 100B rows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordId</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term table, ( T ), about 10M rows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
</tr>
</tbody>
</table>

Figure 1: Basic Schema

A small query language is used. The BNF for the language is as follows

expr: expr and expr
    | expr or expr
    | expr FILTER prop
    | word

prop: prop AND Prop
     | prop OR prop
     | NOT prop
     | NOT expr
     | property

There are seven logical operators defined as shown in Table 1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>e and e</td>
<td>Equijoin with scoring</td>
</tr>
<tr>
<td>e or e</td>
<td>Full outer join with scoring</td>
</tr>
<tr>
<td>e FILTER p</td>
<td>Semijoin: filter e by p</td>
</tr>
<tr>
<td>p AND p</td>
<td>Equijoin without scoring</td>
</tr>
<tr>
<td>p OR p</td>
<td>Full outer join without scoring</td>
</tr>
<tr>
<td>NOT P</td>
<td>s antijoin p (invert the set s)</td>
</tr>
<tr>
<td>NOT e</td>
<td>s antijoin e (invert, omit score)</td>
</tr>
</tbody>
</table>
The query
bay area lang:english
maps to \(((\text{bay AND area}) \text{ FILTER lang:english})\)

The access method uses a sequential scan of a sorted inverted index, which is just a sorted list of all documents that contain a given term. For properties, this is just a sorted list of documents; for expressions the document score is added. All the intermediate results are cached, which serves well since there are no updates.

There are 4 physical operators
\[ \text{OR}(e_1, e_2, e_3, \ldots e_k) \rightarrow \text{expr} \]
Compute the full outer multiway join with scoring.

\[ \text{ORp}(p_1, p_2, p_3, \ldots p_k) \rightarrow \text{prop} \]
Multiway full outer join without scoring.

\[ \text{ANDp}(p_1, p_2, p_3, \ldots p_k) \rightarrow \text{prop} \]
Multiway inner join without scoring.

\[ \text{FILTER}(e_1, e_2, e_3, \ldots e_k)\ (p_1, p_2, p_3, \ldots p_n) \rightarrow \text{expr} \]
Multiway inner join with scoring for the expressions and no scoring for the properties.

The query optimization has three primary tasks: map the logical query, exploit cached results and minimize the number of joins by using large multi-way joins. Top-down optimizer is used as it makes it easy to find the highest cached sub expression. For large multiway join we must look for subsets for terms in cache. We first look for the whole k-way join, then for each k-1 subset, then each k-2 subset, until we get to individual terms. Thus, if \text{FILTER}(a,b,e) and \text{FILTER}(d,f) were in cache (but larger subsets were not), we might map:
\[ \text{FILTER}(a, b, c, d, e, f) \rightarrow \]\
\[ \text{FILTER}(\text{FILTER}(a,b,e), c, \text{FILTER}(d,f)) \]

Other optimization techniques like compression of inverted indices and preloading of cache are also considered.

To understand the paper a person has to know about ‘scoring in search engines’, ‘query plans’ and ‘query optimization’.

Validation Methodology
The validation can be broken into three main parts. First the authors suggest that using a DBMS for search engines is not very effective. For this they cite that they have tried using Informix database for this purpose and it was 10x slower than the hand-built search
engine. This was done in 1996. The authors also say that since then the databases have improved a lot in terms of text search and no recent attempt was made to validate performance of DBMS relative to custom built search engines.

Second the authors state that the design ideas are based on their experience with Inktomi search engine. They have failed to state what all of the design ideas were actually implemented and tested. They haven’t mentioned what kind of searches were done on the Inktomi search engine. Without a good amount of real world searches the design couldn’t be verified.

Third the authors state that they have looked into the approaches of other search engines like AltaVista, Google and Infoseek. But again they have failed to mention the specifics. No comparison has been done on the techniques used by Inktomi versus the other SEs.

Assumptions

The paper only describes score of a document d for query Q as
\[
\text{Score}(Q,d) \equiv \text{Quality}(d) + \sum_{i} \text{Score}(\text{wi}, d)
\]

ie the score of document is sum of scores of individual words in the document and the quality of document. The proximity of words is not considered in the overall score. The schema defined in the paper and the logical query plan also does not cater to word proximity. Word proximity is very important in real world searches. Without considering it in score the documents won’t have the user intended ranking. The word proximity is also language dependent and needs special consideration, in finding the language of the source document.

The authors also assume that search queries are read only and consist of words and properties. We believe that is a valid assumption.

They also assume that each document on the internet has a unique id, based on its url. This might not be true depending how the documents are treated. For example a research paper might be published at many sites. In this case is each published copy considered a separate document or a single document. If separate then the same document may show up in search many times. If same then the assumption is not correct and additional logic might be needed in the indexing code to find such multiple instances.

Another assumption is that documents from one site are evenly distributed across the cluster nodes for load balancing. This might not always be the case and can cause loading of some nodes.

One more reasonable assumption is that discovery and scoring of new documents is done offline and then whole chunks of database are updated. This might be true for many search engines.

Revisions

To improve the paper, we would consider adding information about what was actually implemented, are any of the techniques used in any of the popular search engines. Since most of the search engine revenue comes from advertising, examples how search sensitive search can be added. Also information about logging of searches needs to be added as logging is important to research search pattern and come up with suitable
advertising patterns. Additional information about search on a particular domain like law or image needs to be added. Some discussion can be added about how SEs can use AI to combine all words and figure out the context user is searching for rather than just searching for individual words. These days multimedia data such as images, audio and video have become commonplace on the web. Brief overview about how search of such data can be handled needs to be added to the paper. Some of the related work has been done in [3].

References: