Emergence of web services

Document searching
  * More similar to Information Retrieval than DB Querying
  * “Vertical” searching?

Unprecedented scale

“Deep Web”

Data Integration
  * Schema discovery; schema mapping
  * Federated query processing
Web Services

Tiered architecture
- Web servers interact with clients – serve static pages
- DBMS Backend
- App servers between web servers and DBMS
  - Access the database, maintain the client state
  - Create dynamic webpages etc. . .

Why ?
- Much work on languages for specifying the functionalities, for putting them together etc. . .
HTTP protocol

- “Stateless”
- Applications typically maintain:
  - user name, password, authorizations, resource limits etc
  - ...
- If a DB is used as a backend, need:
  - user name, password, socket, current transaction information, cursor positions...
- Need to simulate state over a stateless protocol
Web Searching

Goal: Answering keyword queries over the WWW

Main Components:

- Crawling and Indexing
- Ranking the documents
- Data structures for efficient execution
- Load balancing/dealing with scale

Why not use DB for this?

Eric Brewer: "Search engines should use DBMS primitives/abstractions, but not SQL/RDBMS"

- Top-down Design
- Data independence
- Declarative languages
Many different algorithms for doing this

From the Brewer paper

Given a single document by itself, \( d \) and a query \( Q = \{[w_1, \cdots, w_k]\} \):

\[
\text{score}(Q, d) = \sum \text{score}(w_i, d)
\]

\[
\text{score}(w_i, d) = \frac{1}{M^e (0.679 \times \log(\text{Freq}(w_i, d)) + 0.223 \times \log(\text{IDF}(w_i)))}
\]

- \( \text{freq}(w_i, d) = \) how many times \( w_i \) appear in \( d \)
- \( \text{IDF}(w_i) = 1/\text{fraction of documents in which the word appears} \)

Web search engines also use “relative positions”
Not good enough - must incorporate additional knowledge

- \( \text{Quality}(d) = \) How important is the document?
- Google uses “pagerank”
- Kleinberg et al. proposed the HITS algorithm around the same time
- Both use the “incoming links” to determine the quality
  - Intuitively, if a large number of high-quality sites link to \( x \), then \( x \) is high quality as well
- Note: The algorithms are executed offline
- Much more sophisticated now

Total Score: Word score + Quality
Efficient Execution

- Originally built in an ad hoc manner
- Brewer: Should have been designed using DB principles
- Schema:
  - Document(DocID, URL, Date, Size, Abstract) (3B rows)
  - WordTable(WordID, DocID, Score, Position Info) (1T)
  - Property(WordID, DocID) (100 billion)
  - Terms(String, WordID, Stats) (10 million)
- A query contains:
  - Properties that must be satisfied
  - Keywords that should be used to rank the documents
  - Arbitrary ANDs, ORs, or NOTs of these
Document table, \( D \), about 3B rows

<table>
<thead>
<tr>
<th>DocId</th>
<th>URL</th>
<th>Date</th>
<th>Size</th>
<th>Abstract</th>
</tr>
</thead>
</table>

Word table, about 1T rows:

<table>
<thead>
<tr>
<th>WordID</th>
<th>DocId</th>
<th>Score</th>
<th>Position Info</th>
</tr>
</thead>
</table>

Property table, about 100B rows:

<table>
<thead>
<tr>
<th>WordID</th>
<th>DocId</th>
</tr>
</thead>
</table>

Term table, \( T \), about 10M rows:

<table>
<thead>
<tr>
<th>String</th>
<th>WordID</th>
<th>Stats</th>
</tr>
</thead>
</table>

Figure 1: Basic Schema

Result Set = \([\text{DocId, Score, URL, Date, Size, Abstract}]\)

\[
\begin{align*}
\text{score} &= \text{Quality}(d) \\
\text{score} &= \sum_{i} \text{Score}(w_i, d)
\end{align*}
\]

Figure 2: The General Query Plan

After finding the set of matching documents and their scores, the \textbf{Top} operator passes up the top \( k \) results (in order) to an equijoin that adds in the document information.
Key data structure
- For each term or property, a list of document IDs that match
- In sorted order (by Document ID)
- Called an “inverted index”

So, the key algorithm to use:
- Sort-merge join

Plan for an all-ANDs query:
- Find the lists for each of the terms and properties
- Sort-merge them together simultaneously

Similarly, ORs and NOTs (treated as properties)
Efficient Execution

- Flattening the query
  - Shallow queries require fewer steps

- Caches
  - Critical, over-riding difference
  - Every result computed during the execution is cached
  - Analogous database question: How to use materialized views
  - Favors a top-down approach for planning
    - Find the largest parts of the query that are already computed
    - Several tricks for handling near-matches
Parallelization is a key
- Google has 100000s of disks and machines

Simple approach
- Partition the large tables by DocIDs
- Small tables (like Terms) are replicated
- Execute the same query plan on all of them
- A master node chooses the query plan

Fault Tolerance?
- Replication-based
Aside: CAP

- Databases ensure ACID (atomicity, consistency, isolation, durability)
- Web search engines don’t care about most of these
- CAP Theorem:
  - Choose two of consistency, availability, and tolerance to partitions
- Databases choose C & P
- Web search engines choose A & P
- BASE: Basically Available, Soft-state, Eventually consistent
Why DB not good for Web Searches

- 10x difference (probably much larger now)
- Several reasons why custom-built search engine better
  - No locking
  - Single hand-optimized query plan
  - Multi-way joins (instead of binary joins)
  - Extensive compression
  - Aggressive caching
  - Careful data representation
  - Hand-written access methods
  - Single address space
  - Not security or access control
Google’s solution to the same problem

Goal: efficient parallelization of various tasks across 1000’s of machines without the user having to worry about the details such as:

- How to parallelize
- How to distribute the data
- How to handle failures
Very Large Scale Distributed (VLSD) Databases

- Bigtable (Google), Dynamo (Amazon), Cassandra (Facebook), PNUTS (Yahoo)
- Similar goals:
  - Distributed, petabyte-scale
  - Transactional (for some definition of transactions)
  - Limited querying
- Bigtable
  - Based on Chubby (Distributed Lock Service)
  - Range partitioning on primary key
  - Support single tuple transactions
- Google Base
Web Data Management Projects

- Richer queries (SIGMOD 2009 paper by Yahoo)
- DMPs at Google (SIGMOD Record Article 2008)
  - Crawling the deep web
  - Searching HTML Tables
  - Bigtable Related:
    - coping with failures
    - sharing machines across users
    - better replication
    - attaching computation to data (like MapReduce)
    - sampling over Bigtables
    - direct support for indexing (see SIGMOD 2009 paper by Yahoo)
Other interesting issues

- Better (page)ranking algorithms
  - Much work in KDD/WWW community

- Information Extraction
  - Trying to extract structured text from unstructured text
  - Some interesting recent work on this + probabilistic databases