Indexing

March 24, 2015

1 Distributed Indexing

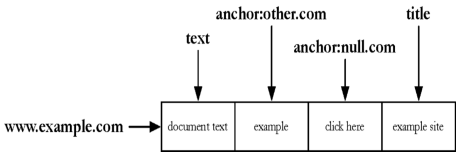
2 Map-Reduce

3 Big Table

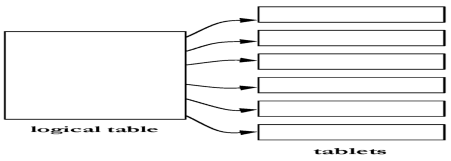
Storage systems such as BigTable are natural fits for distributed algorithm execution. Google invented BigTable to handle its index, document cache, and most of its other massive storage needs. This has produced a whole generation of distributed storage systems, called NoSQL systems. Some examples include MongoDB, Couchbase, etc.

BigTable was developed by Google to manage their storage needs. It is a distributed storage system designed to scale across hundreds of thousands of machines, and to gracefully continue service as machines fail and are replaced. Storage systems such as BigTable are natural fits for processes distributed with MapReduce.

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map." - Chang et al, 2006.



The data in BigTable is logically organized into rows. For instance, the inverted list for a term can be stored in a single row. A single cell is identified by its row key, column, and timestamp. Efficient methods exist for fetching or updating particular groups of cells. Only populated cells consume filesystem space: the storage is inherently sparse.



BigTable rows reside within logical tables, which have pre-defined columns and group records of a particular type. The rows are subdivided into 200MB tablets, which are the fundamental underlying filesystem blocks. Tablets and transaction logs are replicated to several machines in case of failure. If a machine fails, another server can immediately read the tablet data and transaction log with virtually no downtime.

All operations on a BigTable are row-based operations. Most SQL operations are impossible here: no joins or other structured queries. BigTable rows can have massive numbers of columns, and individual cells can contain large amounts of data. For instance, it's no problem to store a translation of a document into many languages, each in its own column of the same row.

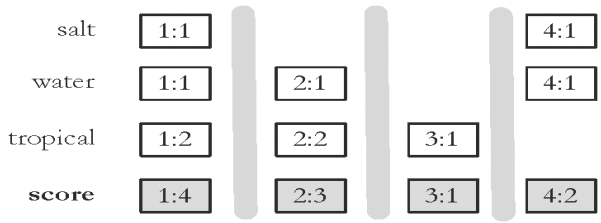
4 Query Processing

There are two main approaches to scoring documents for a query on an inverted index.

· **Document-at-a-time** : It processes all the terms' posting lists in parallel, calculating the score for each document as it's encountered.

· **Term-at-a-time** : Term-at-a-time processes posting lists one at a time, updating the scores for the documents for each new query term. There are optimization strategies for either approach that significantly reduce query processing time document.

4.1 Doc-at-a-Time Processing



We scan through the postings for all terms simultaneously, calculating the score for each document. We remember scores for the top k documents found so far.

Recall that the document score has the form:

$\sum_{w \in q} f(w).g(w)$

for document features f(w) and query features g(w).

This algorithm implements doc-at-atime retrieval. It uses a list L of inverted lists for the query terms, and processes each document in sequence until all have been scored. The documents are placed into the priority queue R so the top k can be returned

4.2 Term-at-a-Time Processing

For term-at-a-time processing, we read one inverted list at a time. We maintain partial scores for the documents we've seen so far, and update them for each term. This may involve remembering more document scores, because we don't necessarily know which documents will be in the top k (but sometimes we can guess).