Web-scale Entity-Relation Search Architecture

Soumen Chakrabarti
IIT Bombay
soumen@cse.iitb.ac.in

Devshree Sane
IIT Bombay
devshree.sane@gmail.com

Ganesh Ramakrishnan
IIT Bombay
ganesh@cse.iitb.ac.in

ABSTRACT

Enabling entity search and ranking at Web-scale is fraught with many challenges: annotating the corpus with entities and types, query language design, index design, query processing logic, and answer consolidation. We describe a Web-scale entity search engine we are building to handle over a billion Web pages, over 200,000 types, over 1,500,000 entities, and hundreds of entity annotations per page. We describe the design of compressed, token span oriented indices for entity and type annotations. Our prototype demonstrates the practicality of Web-scale entity-relation search.

Categories and Subject Descriptors:
H.3.3 [Information Search and Retrieval]: Search process, H.3.4 [Systems and Software]: Performance evaluation (efficiency and effectiveness)
General Terms: Measurement, Performance
Keywords: Entity-relation search, Web-scale, Index design

1. INTRODUCTION

Semantic retrieval, or retrieving fine-grained artifacts on the Web like people, organizations, epochs, etc., has been gaining popularity in recent years. However, most prior work like expert search [1], factoid question answering [3] or entity search [5] is focused on retrieval quality, and overlooks issues like index design, query processing, and scalability. Notable exceptions like SSQ [6] have applied their systems to a modest size corpus of 2.4 million documents and only 10 types. Consequently, very little public-domain knowledge is available on the engineering aspects of designing and implementing large-scale semantic search systems.

Our aim is to build a Web-scale open domain search system (CSAW[1]) that enables semi-structured search over a corpus annotated with entities and types from large catalogs. We are building a system that scales with increasing catalog size and entity annotation density without significantly bloating the required index space or sacrificing query processing speeds.

Query language. We designed a semi-structured query language that draws on many recent efforts [4, 5, 6, 2]. Entity search queries are just a subset of the queries we support. The query language lets users name things by binding a variable x to entities e of type T, e.g., $x \in^+$ *Physicist*. Keywords and entities along with ordering and proximity constraints form a *ContextQuery*. E.g., to identify the volcano that destroyed the city of Pompeii, one might use the query `Unordered(30;` $t \in^+$ `Type:Volcano, destroy, +City:Pompeii)`. Full syntax will be published[1].

Automatic annotations. Associating a sequence of tokens to one or more entities in the catalog is called *annotation.* Most approaches to annotation are high-precision, low-recall, sparse and restricted to very few (10–20) types. However, redundancy is the key to accuracy in the face of noisy automatic and/or social annotations. Hence, high-recall annotation is critical, motivating the need for a space-efficient index.

Retrieval, Aggregation and Ranking. A *witness* is a span of tokens that satisfies a ContextQuery. Witnesses are turned into feature vectors, scored and aggregated. If a standard inverted index were used to look for witnesses, we would also need to consult the so-called "forward index" (compressed corpus) to build feature vectors, which is very time consuming due to random disk seeks and decompression. Accessing the forward index is impractical for our case, since creating a ranked list of a few entities may require aggregating over millions of witnesses.

Indexing. A standard inverted index lacks many essential features that we need. **1. Efficient non-redundant storage policy:** Entity and type mentions can span multiple tokens (like "New York"). Replicating the entity at every token position leads to distorted proximity measures and document and corpus frequencies. **2. Handling disparate overlapping annotations:** The automatic annotator might output multiple or overlapping annotations for the same span with different confidence scores; e.g. entities Apple_Inc. and New_York_City might both be associated with span "Big Apple". **3. Storing arbitrary information for scoring:** Beyond confidence scores, we wish to inline more sophisticated scoring and ranking information. Our SIP index (described next) overcomes these shortcomings by packing more information in the index by fine-grained **I**nterleaving of **S**nippets in **P**osting lists.

2. THE SIP INDEX

We could find no public-domain indexing package that was adequate for our purpose. Lemur-Indri can directly index annotations but is limited to around 4000 distinct entities. Lucene's per position payload can only be an integral number of bytes, leading to inefficient compression. As the catalog and annotators evolve, some of our indices will need to be revised faster than others. This is not manageable in Lucene because it combines indices for all fields into a single set of files. Separating the fields into different indices is dangerous, because Lucene gives no guarantees that document IDs will remain immutable. In contrast, MG4J has immutable document IDs, field-specific separate index files and is very modular. We reuse many of its lower IO level modules for our implementation.

Native span support. Each document block in a SIP posting list for an entity contains

1. Gap gamma coded document ID (where gap is the difference between this and the previous document ID)
2. Number of occurrences of the entity in the document.

[1]`http://www.cse.iitb.ac.in/soumen/doc/CSAW/`

Copyright is held by the author/owner(s).
WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

3. Sequence of intervals in order of non-decreasing left end points. Each interval (l, r) is encoded as
   (a) the left endpoint l, gap gamma coded wrt the left endpoint of previous interval, and
   (b) the right endpoint r, gap-gamma coded wrt to l.

Inlining approach. We inline the entity IDs as 32 bit integers into the posting lists. These entity IDs are called as *long* entity IDs and uniquely identify an entity in the catalog. A document block in a SIP type posting list contains entity IDs inlined with each occurrence.

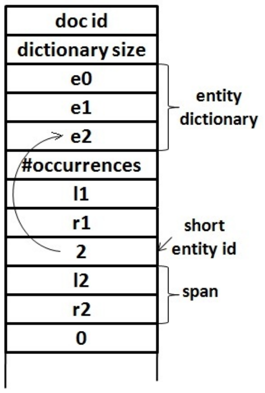

Figure 1: SIP document block with entity dictionary

Dictionary based approach. Inlining comes at a space premium. Observe that, in any given document, the set of mentioned entities e belonging to a type T is small. Exploiting this, we add a *local entity dictionary* to each document block in the SIP type posting list. The entity dictionary (Figure 1) is a list of long entity IDs in decreasing order of occurrence frequencies in that document. We assign *short* entity IDs, starting from 0, representing the relative position of the entity in the dictionary. The short entity IDs are gamma-coded and inlined.

Query Processing. A query is compiled into an abstract syntax tree. Each query node in the AST outputs a list of spans. For leaf nodes in the AST, spans can either be degenerate 1-token spans or general annotation spans. Internal nodes merge the spans and, if the query constraints are satisfied, percolate these spans up the tree to form witnesses. The witness contains all the information needed to create the feature vector, without involving a disk seek.

3. EXPERIMENTS

Our testbed consists of forty HP DL160G5 servers, with 2.2GHz 8-core Xeon processors and 8GB RAM. All servers use Ubuntu 10.04 LTS and Sun Java 1.6.0_18. Our entire system has been implemented in Java.

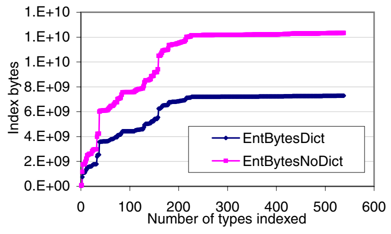

Figure 2: Dictionary uses half the bits to encode entity bindings

The average per-document entity dictionary size is less than 3 giving us significant compression. Figure 2 shows that using the entity dictionary almost halves the space needed by the entity payload in the SIP type index. Figure 3 shows that the overhead of decompression of intervals and entity IDs is minimal. For a full index stripe of 50 million documents, the weighted average scan time for types is only 2.6 seconds; a modest multiple of 0.4 seconds for words.

We implemented SSQ's two part (TEPL+EDPL; see their paper for details) indexing strategy that has the potential

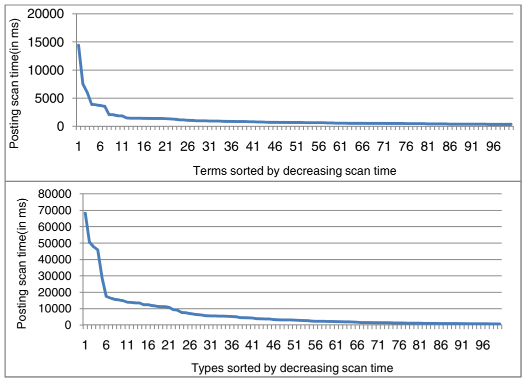

Figure 3: Comparative scan times for word and SIP postings, sorted in decreasing order of scan time

to speed up query execution, but at the cost of substantially larger space for the index.

For experiments, volunteers converted 913 TREC and INEX queries to queries in our query language, resulting in 617 unique types, that were indexed for the experiments. Using index selection techniques similar to IR4QA [4], we can create a SIP index for thousands of types with only a modest hit on query processing time. We used a slice of our Web corpus which was annotated to varying densities. Figure 4 shows the index sizes obtained for SIP and SSQ.

Density	0.068	0.150	0.299	0.438
SSQ-TEPL	9.693	9.510	20.160	33.505
SSQ-EDPL	8.417	8.567	8.840	8.945
SSQ-Total	18.110	18.077	29.000	42.450
SIP entity	0.226	0.223	0.448	0.689
SIP type	0.290	0.285	0.658	0.828
Our total	0.516	0.508	1.106	1.517

Figure 4: Index space (GB) at varying annotation densities

SSQ-EDPL is unaffected by annotation density, while SSQ-TEPL explodes with increasing density. For all densities, SIP provides significantly smaller index. The average query processing time for SIP using 913 queries and 50 million documents was only 3.21 seconds. These encouraging results show that Web-scale entity search is practical. We are currently exploring alternate SIP index designs and query processing algorithms.

4. REFERENCES

[1] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Information Processing and Management*, 45(1):1–19, 2009.
[2] S. Banerjee, S. Chakrabarti, and G. Ramakrishnan. Learning to rank for quantity consensus queries. In *SIGIR Conference*, 2009.
[3] J. Bian, Y. Liu, E. Agichtein, and H. Zha. Finding the right facts in the crowd: Factoid question answering over social media. In *WWW Conference*, 2008.
[4] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW Conference*, Edinburgh, May 2006.
[5] T. Cheng, X. Yan, and K. C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB Conference*, pages 387–398, Sept. 2007.
[6] X. Li, C. Li, and C. Yu. EntityEngine: Answering entity-relationship queries using shallow semantics. In *CIKM*, Oct. 2010. (demo).