Data Management Projects at Google

Michael Cafarella Edward Chang Andrew Fikes Alon Halevy Wilson Hsieh
Alberto Lerner Jayant Madhavan S. Muthukrishnan

1. INTRODUCTION

This article describes some of the ongoing research projects related to structured data management at Google today. The organization of Google encourages research scientists to work closely with engineering teams. As a result, the research projects tend to be motivated by real needs faced by Google's products and services, and solutions are put into production and tested rapidly. In addition, because of the sheer scale at which Google operates, the engineering challenges faced by Google's services often require research innovations.

In Google's early days, structured data management was mostly needed for storing and serving data related to ads. However, as the company grows into hosted applications and the analyses performed on its query streams and indexes get more sophisticated, structured data management is becoming a key infrastructure in all parts of the company.

What we describe below is a subset of ongoing projects, not a comprehensive list. Likewise, there are others who are involved in structured data management projects, or have contributed to the ones described here, some of whom are Roberto Bayardo, Omar Benjelloun, Vignesh Ganapathy, Yossi Matias, Rob Pike and Ramakrishnan Srikant.

Sections 2 and 3 describe projects whose goal is to enable search on collections of structured data that exist today on the web. Section 2 describes our efforts to crawl content that resides behind forms on the web, and Section 3 describes our initial work on enabling search on collections of HTML tables. Section 4 describes work on mining large collections of data and social graphs. Sections 5 and 6 describe recent progress on BigTable, our main infrastructure for storing structured data.

2. CRAWLING THE DEEP WEB

Jayant Madhavan and Alon Halevy

The Deep Web refers to content hidden behind HTML forms. In order to get to a web page from the Deep Web, a user has to perform a form submission with valid input values in the form's fields. Since web crawlers primarily rely on hyperlinks to discover web pages, they are unable to reach pages on the Deep Web that are subsequently not indexed by search engines. The Deep Web has been acknowledged as a significant gap in the coverage of search engines and various accounts have hypothesized the Deep Web to have an much more data than the currently searchable World Wide Web. Included in the Deep Web are a large number of high quality sites, such as store locators and government sites. Hence, we would like to extend the coverage of the Google search engine to include web pages from the Deep Web.

There are two complementary approaches to offering access to Deep Web content. The first approach, essentially a data integration solution, is to create vertical search engines for specific domains (e.g., cars, books, real-estate). In this approach we could create a mediator form for the domain at hand and semantic mappings between individual data sources and the mediator form. At web-scale, this approach suffers from several drawbacks. First, the cost of building and maintaining the mediator forms and the mappings is high. Second, identifying the domain, and the forms within the domain, that are relevant to a keyword query is extremely challenging. Finally, data on the web is about everything and domain boundaries are not clearly definable, not to mention the many different languages – creating a mediated schema of everything will be an epic challenge.

The second approach, sometimes called the surfacing approach, pre-computes the most relevant form submissions for all interesting HTML forms. The URLs resulting from these submissions can then be indexed like any other HTML page. Importantly, this approach enables leveraging the existing search engine infrastructure and hence the seamless inclusion of Deep Web pages into web search results and this leads us to prefer the surfacing approach. We note that our goal is to drive new traffic to Deep Web sites that until now were visited only if people know about the form or if the form itself came up in a search result. Consequently, it is not crucial that we obtain all the possible form submission from these sites, but just enough to drive more traffic. Furthermore, our pre-computed form submissions function like a seed to unlocking the site – once an initial set of pages are in the index, the crawling system will automatically leverage the internal structure of the site to discover other pages of interest.

We have developed a surfacing system at Google that has already enhanced the coverage of our search index to include web pages from over a million HTML forms. We are sub-

sequently able to drive over a thousand queries per second from the Google.com search page to Deep Web content.

In deploying our solution, we had to overcome several challenges. First, a large number of forms have text box inputs and require valid inputs values to be submitted. Therefore, the system needs to choose a good set of values to submit in order to surface the most useful result pages. We use a combination of two approaches to address this challenge. For search boxes, which accept most keywords, we predict good candidate keywords by analyzing the content of already indexed pages of the website. For typed text boxes, that only accept a well-defined set of values, we attempt to match the type of the text box against a library of types that are extremely common across domains, e.g., zip codes in the US.

Second, HTML forms typically have more than one input and hence a naive strategy of enumerating the entire Cartesian product of all possible inputs can result in a very large number of URLs being generated. Crawling too many URLs will drain the resources of a search engine web crawler while also posing an unreasonable load on web servers hosting the HTML forms. Interestingly, when the Cartesian product is very large, it is likely that a large number of the form submissions result in empty result sets that are useless from an indexing standpoint. For example, the search form on cars.com has 5 inputs and a Cartesian product will yield over 200 million URLs, even though cars.com has only 650,000 cars on sale. We have developed our algorithm that intelligently traverses the search space of possible form submissions to identify only the subset of input combinations that are likely to be useful to the search engine index. On average, we only generate a few hundred form submissions per form. Furthermore, we believe the number of form submissions we generate is proportional to the size of the database underlying the form site, rather than the number of inputs and input combinations in the form.

Third, our solutions must scale and be domain independent. There are millions of potentially useful forms on the web. Given a particular form, it might be possible for a human expert to determine through laborious analysis the best possible submissions for that form, but such a solution would not scale. Our goal was to find a completely automated solution that can be applied to any web form in any language or domain. To date, our system has crawled over a million forms in over 50 languages and in hundreds of domains.

We note that we only index informational form sites. We take precautions to avoid any form that requires any personal information or is likely to have side effects. For example, we do not analyze forms that use the `post` method, have `password` or `textarea` inputs, or include keywords such as username, login, etc.

While our surfacing approach has generated considerable traffic, there remains a large number of forms that continue to present a significant challenge to automatic analysis. For example, many forms invoke Javascript events in `onselect` and `onsubmit` tags that enable the execution of arbitrary Javascript code, a stumbling block to automatic analysis. Further, many forms involve inter-related inputs and accessing the sites involve correctly (and automatically) identifying their underlying dependencies. Addressing these and other such challenges efficiently on the scale of millions is part of our continuing effort to make the contents of the Deep Web more accessible to search engine users. Finally, we also note that *site maps* are another mechanism that allows the content providers to give lists of URLs in XML files to search engines, and therefore expose content behind from the deep web. All major search engines today support the site maps protocol described in www.sitemaps.org. The content provided by site maps tends to be complimentary to the content that is automatically crawled using the techniques described above.

3. SEARCHING HTML TABLES

Michael Cafarrela and Alon Halevy

The World-Wide Web consists of a huge number of unstructured hypertext documents, but it also contains structured data in the form of HTML tables. Some of these tables contain relational-style data, with tuple-oriented rows and a schema that is implicit but often obvious to a human observer. Indeed, these HTML tables make up the largest corpus of relational databases that we are aware of (numbering more than 100M databases, with more than 2M unique schemas). The WebTables project is an effort to extract high-quality relations from the raw HTML tables, to make these databases queryable, and to use this unique dataset to build novel database applications.

The first WebTables task is to recover high-quality relations from a general web crawl. Relation recovery includes two steps: first, WebTables filters out raw HTML tables that do not carry relational data (such as those used for page layout). Second, for those tables that pass the relational filter, WebTables recovers schema metadata such as column types and labels. There is no way for an HTML author to reliably indicate whether a table is relational, or to formally declare relational metadata. Instead, WebTables must rely on a host of implicit hints. For example, tables that are used for page layout will often contain very long and text-heavy cells, whereas tables for relational data will usually contain shorter cells of roughly consistent length. Similarly, one way to test whether a table author has inserted a "header row" is to see if the first row is all strings, with different types in the remaining rows.

The second WebTables challenge is to design a query tool that gives easy access to more than a hundred million unique databases with more than 2 million unique schemas. We have built a "structured data search engine" in which the user types a search-style text query, and the engine returns a relevance-ranked list of databases instead of a list of URLs. After the user has chosen a relevant database, she can apply more traditional structured query tools (such as selection, projection, etc). Additionally, the engine can automatically apply certain structured operations without waiting for the user. For example, WebTables can examine the contents of a table and try to generate a visualization that is domain-appropriate and "interesting," displaying it next to the table in query search results.

Finally, WebTables uses the corpus of recovered databases to build a series of new applications. We have designed two so far. The first is *schema autocomplete*, in which a user enters one or more desirable data attributes (e.g., "name") and the autocompletor suggests the rest (e.g., "address", "city", "zip", "phone", etc.). The second is *synonym finding*, a tool that automatically computes which table attributes appear to be synonymous (e.g., "song" and "title", "telephone" and "tel-#"). This data can then be used to improve schema matching. Both tools are made possible by attribute-label

co-occurrence statistics derived from the corpus of recovered databases.

WebTables works today (available only internally), but we believe there are many future research questions. We can improve the performance of existing steps (such as relation recovery accuracy and database ranking quality), expand the input data beyond simple HTML tables (perhaps including HTML lists or Excel spreadsheets), and build new applications on the recovered data (such as data-suggestion, a "vertical" analog to schema autocompletion). There are also a host of questions prompted by a "data-centric" view of the web: we are currently researching whether it is possible to automatically find joins between structured data recovered from different web pages. For example, to find where world leaders reside, we might join a table of countries and capital cities to a table of countries and their premiers.

The WebTables Project and the Deep-Web Crawl Project are parts of our larger research effort into dataspaces [6], and on data integration with uncertainty as basis for building dataspace systems. Some of our earlier work in this area is described in [4, 5, 8].

4. LARGE-SCALE DATA MINING AND COMMUNITY PRODUCTS

Edward Chang

We now describe our work on developing scalable algorithms for mining large-scale Web data and social graphs. This work is lead by Edward Chang, who heads Google Research in China. Building upon this scalable data mining infrastructure, the engineering team developed and launched two social-network products, and drastically reduced pagerank spam rate in China (from 5% in 2006 to now under 1%).

The research work focused on parallelizing six mission-critical machine learning algorithms including Support Vector Machines (SVMs), Singular Vector Decomposition (SVD), Spectral Clustering, Association Mining, Probabilistic Latent Semantic Analysis (PLSA), and Latent Dirichlet Allocation (LDA) to take advantage of Google's massive, distributed storage and computing services. In particular, his team parallelized SVMs [1], and made the code publicly available through Apache open source.

SVMs are widely used for classification tasks due to their strong theoretical foundation and empirical successes. Unfortunately, SVMs suffer from scalability problems in memory use and computational time. We developed parallel SVM algorithm (PSVM) to remedy these problems. PSVM reduces memory use by performing a row-based approximate matrix factorization, and by loading only essential data to each of the parallel machines. PSVM reduces computation time by intelligently reordering computation sequences and by performing them on parallel machines. Furthermore, PSVM supports fault-tolerant computing to recover from computer-node failures.

In terms of computational complexity, let n denote the number of training instances, p the reduced matrix dimension after factorization (p is significantly smaller than n), and m the number of machines. PSVM reduces the memory required by the Interior Point Method (IPM) from $O(n^2)$ to $O(np/m)$, and improves computation time to $O(np^2/m)$. For instance, a task taking 7-days to run on one single machine takes PVSM to complete in two hours on 200 machines. PSVM is currently used internally at Google for identifying spammy and objectionable Web sites. Since PSVM was made publicly available, the code has been widely downloaded.

Besides PSVM, the parallel version of SVD, PLSA, and LDA has also been made available at Google internally. These algorithms are useful for tasks of classification and collaborative filtering. For classification, PLSA is employed to provide tags for user questions, short messages, and user posts. For collaborative filtering, PLSA and LDA are used to assist various recommendation features, e.g., friend/expert suggestion, forum recommendation, and ads matching. Together, these algorithms power two products which we describe next.

The first product is Knowledge Search, which was first launched in Russia and then China [12], and is now being launched in several other countries. Knowledge Search allows users to post questions and then matches experts to timely answer questions. The distinguishing feature of this product compared to competing products is that it offers online question classification, related-question recommendation, and topic-sensitive expert matching. All of these features are empowered by the aforementioned machine-learning infrastructure.

The second product, Laiba, is a social-network product initially developed based on Orkut. We first localized the product, and then quickly expanded its features such as photo sharing and user-interaction services. We launched Laiba in China in 2007 [10]. Similar to Knowledge Search, this product is supported by large-scale data mining algorithms to support friend/community/content recommendation. The team is now further expanding Laiba to support the Google Open-Social platform [11] that will enable third-party applications to plug Laiba and other social-networks.

5. BIGTABLE

Andrew Fikes and Wilson Hsieh

We have built a system called Bigtable [2] to store structured or semi-structured data at Google. (The Google File System [7] is used when a file-system interface is acceptable.) Bigtable can be viewed at a systems level as a distributed, non-relational database; at the algorithmic level as a highly distributed multi-level map; or at the implementation level as a variant of a distributed B-tree. We use Bigtable to store data for many different projects, such as web indexing, Google Earth, and Orkut.

Bigtable has been under active development since late 2003, and its first deployment in production was in mid-2005. Over the last few years, the deployment of Bigtable has grown steadily. As of January 2008, there are more than 600 Bigtable clusters at Google; the largest cluster has over 2000 machines. The largest cells store over 700TB of data, and the busiest cells sustain 100K operations/second.

Besides our day-to-day "maintenance" work (improving the performance of the system, fixing bugs, training users, writing documentation, improving manageability, etc.), we are still adding new features to Bigtable. In addition, we continue to redesign parts of the system as our users run larger Bigtable clusters. Following is a brief description of some of the higher-level issues that we are working on:

- `Coping with failure.` Bigtable software runs on lots of machines: enough to almost guarantee that we will

eventually run into buggy hardware (faulty memory is one of the more problematic issues). We are investigating better ways to deal with such problems, without hurting performance dramatically.

- `Sharing machines across users.` Although Bigtable's interface supports multiple users, the implementation did not until recently do a good job of providing sufficient isolation between them. We are still improving Bigtable's ability to do resource management and isolation.

- `Cross-data-center replication.` Bigtable currently allows clients can set up lazy replication between their tables (which can be in different data centers). This replication system guarantees eventual consistency, which suffices for many of our clients. However, some clients (in particular, those that are building user-facing products) need stronger guarantees on the consistency and availability of their data. We are building in support for these stronger consistency guarantees, both on top of Bigtable and inside Bigtable.

- `Attaching computation to data.` To support long-running computations that need to access data in Bigtable, we have been adding APIs that allow clients to run code on the same machines as their data. Although the Map-Reduce framework [3] does provide some support for running computation near data, it does not provide any strong guarantees.

- `More expressive queries.` Bigtable does not support SQL; it currently supports the use of a Google-designed language called Sawzall [9] to describe server-side filtering of data. For various reasons, this support is awkward to use, and requires a fair amount of work to describe simple filters. We are in the process of implementing a small language that will support the most common kinds of filters that our clients need.

- `Direct support for indexing.` Many of our clients want to store indexed data in Bigtable. Currently, they have to manage the indices themselves. We are in the process of building support for indices directly into Bigtable.

Most of these features are being added directly to Bigtable, but some features are being built as client layers on top of Bigtable. The Megastore project, for example, is building more general support for transactions, consistent replication, and DAO. Although Bigtable is not a database, most of the features that we are adding are very familiar to the database community. That fact is unsurprising, given the usefulness of these features. What will be interesting is what the design and implementation of Bigtable is in 1-2 years, and what it tells us about building high-performance, widely distributed data-storage systems.

6. MINITABLES: SAMPLING BIGTABLE

Alberto Lerner and S. Muthukrishnan

As described above, Bigtable is a high-performance, distributed, row-storage system that is highly scalable, but it is not meant to provide relational query processing or sophisticated indexing. Therefore, some accesses to a Bigtable may involve large parallel scans. Although Google's infrastructure supports these scans relatively well, there are instances where it is desirable to work with a *sample* of the data in a Bigtable. This section discusses the challenges and opportunities to build such a sampling feature.

A row in a Bigtable is keyed by a unique string called a rowname and each row has its data spread across a number of column families. A column family may comprise a variable number of actual columns. Since Bigtables are sparse structures, a row may or may not exist for a given query, depending on which columns that query requested. Data is maintained in lexicographical order but different columns may or may not be stored apart. Because of such semantics and storing scheme, skipping N rows is not feasible without actually reading them. Even finding the count of rows in a Bigtable at any point in time can be done only probabilistically. On the bright side, since Bigtable does not provide a relational query engine, we do not need to consider what are suitable sampling methods for various relational operators (like joins) or take into account how sampling errors compound with increasing levels of query composition.

Uniform Random Sampling. Our sampling scheme extracts and presents a sample of a Bigtable's rows as if it were a Bigtable itself, which we call a *Minitable*. The rationale here is that code written to run against a Bigtable can run unchanged against a sample thereof.

Our sampling is based on a hash scheme. We pick a convenient hash function that maps the rowname space into a very large keyspace (e.g., a $ax + b \mod p$ function, where p is as large as 2^{128}). The rows falling into the first fp keys where f is the relative sample size (it is a fraction), would belong in the sample. Formally, we pick a hash function $h : R- > 0..p$ and if $h(r) \in [0, fp - 1]$, then add r to the sample. It is easy to see that the expected size of the sample is $f * 100\%$ of the Bigtable rowcount independent of the rowcount, and the probability that a particular row r is in the sample is f, as desired. This hash-based sampling method supports maintenance of the sample with each Bigtable *mutation* (insert, update, or deletion).

Only the system may forward relevant mutations from the Bigtable to the Minitable. Otherwise, the latter would behave as just any other Bigtable: it could be backed up and even be replicated. We are currently deploying Minitables in the repository of documents that the crawling system generates. Several Minitables, each with a different sample factor, allow that system to compute aggregates much faster and moer often.

Biased Sampling. Uniform random sampling is quite useful but some scenarios require biased sampling methods. We are currently working on one such extension that we call *Mask Sampling*. In this scheme, the decision to select a row to the sample is still based on its rowname but now a user may specify a *mask* m over it. The mask, which can be a regular expression that matches portions of a rowname, is used to group rows together. Two rows belong to a same group if their masks result in the same string. Mask sampling guarantees that if a group is selected to the sample, that group will be adequately represented there, regardless of that group's relative size.

A typical application would be over a Bigtable that stores web pages keyed by their URL's. If one used uniform ran-

dom sampling over it, one may lose information about domains with relatively few pages. With mask sampling, one can define how to extract a domain from a URL and determine that each domain has the same probability to appear in the sample.

Specifically, our procedure should return a possibly non-uniform sample of $\langle k.m \rangle$, that is, rowname k projected only on the mask. There are at least two details that make the problem interesting. (a) The set of distinct $\langle k.m \rangle$'s may be large and need to be sampled. Using our previous example, there may be simply too many domains to fit in the desired sample size. (b) Even though the rownames are unique, the set of $\langle k.m \rangle$'s is often not: for each $\langle k.m \rangle$ value, we have a set of rows from the Bigtable and we need to determine what to retain in the Minitable. Again, using the example URL table, we may need to sample within a chosen domain. Let us consider the set $S(n)$ of rows that have $\langle k.m \rangle = n$. Then, ideally, we would like keep all rows r from $S(n)$ if $|S(n)|$ is small, to sub-sample with moderate probability if $|S(n)|$ is larger and more aggressively when $|S(n)|$ is huge.

The hash-sampling procedure generalizes to the biased case as well. We have $h_1 : \langle k.m \rangle \rightarrow [0 \ldots p]$ and retain those that hash into the first f fraction of the range, as before. Then, within each $\langle k.m \rangle = n$ that is retained by h_1, we apply h_2 (dependent on n), this time on the entire rowname as opposed to just the mask. Here, we assume that we have a side table $T : \langle k.m \rangle = n \rightarrow g_n$, which is often programmed by an offline logic or is easy to maintain in a lazy manner in practice. (It can be indirectly obtained if a uniform Minitable is present.) We call this table *gtable* because it contains a row for each groupby specified by the mask.

In practice, this sampling scheme may give us a biased Minitable from the URL Bigtable with a representative sample of domains. Each of them would carry enough rows to allow for the computation of approximate aggregations, for instance, even if the domains chosen had a large variance in term of number of rows in the base Bigtable.

7. REFERENCES

[1] E. Y. Chang, K.Zhu et al., Parallelizing Support Vector Machines on Distributed Computers. Proceedings of NIPS 2007. downloadable open source at http://code.google.com/p/psvm/.

[2] Chang, F., et al. Bigtable: A Distributed Storage System for Structured Data. In *Proc. of the 7th OSDI* (Dec. 2006), pp. 205–218.

[3] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *Proc. of the 6th OSDI* (Dec. 2004), pp. 137–150.

[4] Dong X. and Halevy A. Indexing Dataspaces. Proceedings of the International Conference on Management of Data (SIGMOD), pp. 43-54, 2007.

[5] Dong X., Halevy A., and Yu C. Data Integration with Uncertainty. International Conference on Very Large Databases (VLDB), pp. 687-698, 2007.

[6] Franklin M., Halevy A., and Maier D. From databases to dataspaces: a new abstraction for information management. SIGMOD Record, 34(4): 27-33, 2005.

[7] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. In *Proc. of the 19th ACM SOSP* (Dec. 2003), pp. 29–43.

[8] Madhavan J., Cohen S., Dong X., Halevy A., Jeffery S., Ko D., and Yu C. Web-Scale Data Integration: You can only afford to Pay as You Go. Proceedings of CIDR, pp. 342-350, 2007.

[9] Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming Journal 13*, 4 (2005), 227–298.

[10] http://liaba.tianya.cn.

[11] Google Open Social. http://code.google.com/apis/opensocial.

[12] http://otvety.google.ru/otvety/. http://wenda.tianya.cn.