Web-scale Data Integration: You can only afford to Pay As You Go

Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong,
David Ko, Cong Yu, Alon Halevy
Google, Inc.
jayant@google.com, jeffery@cs.berkeley.edu, shirleyc@cis.upenn.edu, lunadong@cs.washington.edu,
dko@google.com, congy@eecs.umich.edu, halevy@google.com

ABSTRACT

The World Wide Web is witnessing an increase in the amount of structured content – vast heterogeneous collections of structured data are on the rise due to the Deep Web, annotation schemes like Flickr, and sites like Google Base. While this phenomenon is creating an opportunity for structured data management, dealing with heterogeneity on the web-scale presents many new challenges. In this paper, we highlight these challenges in two scenarios – the Deep Web and Google Base. We contend that traditional data integration techniques are no longer valid in the face of such heterogeneity and scale. We propose a new data integration architecture, PAYGO, which is inspired by the concept of dataspaces and emphasizes pay-as-you-go data management as means for achieving web-scale data integration.

1. INTRODUCTION

Since its inception, the World Wide Web has been dominated by unstructured content, and searching the web has primarily been based on techniques from Information Retrieval. Recently, however, we are witnessing an increase both in the amount of structured data on the web and in the diversity of the structures in which these data are stored. The prime example of such data is the *deep web*, referring to content on the web that is stored in databases and served by querying HTML forms. More recent examples of structure are a variety of annotation schemes (e.g., Flickr [14], the ESP game [33], Google Co-op [16]) that enable people to add labels to content (pages and images) on the web, and Google Base [17], a service that allows users to load structured data from any domain they desire into a central repository.

A common characteristic of these collections of structured data is that they yield heterogeneity at scales unseen before. For example, the deep web contains millions of HTML forms with small and very diverse schemata, Google Base contains millions of data items with a high degree of diversity in their structures, and Google Co-op is producing large collections of heterogeneous annotations. Heterogeneity in this environment is reflected in two aspects: First, the same domain can be described using multiple different schemata (e.g., multiple Google Base schemata describing vehicles); Second, there may be many ways to describe the same real-world entity (e.g., multiple ways of referring to the same product or person).

The presence of vast heterogeneous collections of structured data poses one of the greatest challenges to web search today. To take a concrete example, suppose a user poses the query "Honda Civic" to a web-search engine. We would like the engine to return (and properly rank) results that include, in addition to unstructured documents, links to web forms where the user can find new or used cars for sale, links to sites where car reviews can be found, entries from Google Base that may be relevant, and links to special sites that have been annotated by car enthusiasts as relevant. If the user further specifies a geographic location with the query, the engine should specialize the results appropriately, and perhaps include links to Honda dealers in the area, or a link to an appropriate dealer locater form.

Improving search in the presence of such heterogeneous data on the web leads to a fundamental question: *are too many structures (i.e., schemata) akin to no structure at all?* In other words, are we doomed to query this content with traditional web-search techniques based on Information Retrieval? Or can we extend techniques from data management, in particular from heterogeneous data integration, to improve search in such contexts?

This paper contends that traditional data integration techniques are no longer valid in the face of such heterogeneity and scale. Thus, we propose a new data integration architecture, the PAYGO[1] architecture, as a methodology for approaching this challenge. The PAYGO architecture is inspired by the concept of dataspaces [15] that emphasizes pay-as-you-go data management.

We begin by describing two data management efforts at Google in some detail, and use them to expose the challenges faced by web-scale heterogeneity. First, we discuss our work on searching the deep web. We describe how the scale of the problem affects two alternative approaches to deep-web querying: *run-time query reformulation* and *deep-web surfacing*. The former approach leaves the data at the sources and routes queries to appropriate forms, while the latter attempts to add content from the deep web to the web index. We also describe the first study of the deep web that is based on a commercial index; the study suggests that the deep web contains millions of sources. Second, we consider Google Base and describe how schema when available can be used to enhance a user's search experience. This improvement, however, comes at the expense of large-scale heterogeneity that arises naturally as a result of the large numbers of independent contributions of structured data to Google Base. Additionally, we touch briefly upon annotation schemes to further illustrate the challenges and opportunities of structured data on the web.

[1]PAYGO is an acronym for Pay-As-You-GO, the key tenet of this architecture

Two principles arise from our experiences. First, web-scale heterogeneity is not about modeling a domain or set of domains. On the web, we need to model *everything*. Therefore, we cannot expect to address the challenges with a system that is based on a well-designed schema. Second, web-scale integration is a pay-as-you-go, ongoing process. It starts with little semantic glue, and uses a set of mechanisms to continuously improve the integration over time. Hence, web-scale data integration architecture needs to include the mechanisms for supporting this process.

Based on these principles, we describe the PAYGO data integration architecture. In PAYGO, there is no single mediated schema over which users pose queries. Instead, there are sets of schemata that are clustered into topics. Semantic mappings between sources, the core of data integration systems, are typically approximate. Queries are typically posed as keywords, respecting the main search paradigm on the Web, and are *routed* to the relevant sources. To support all of the above, a PAYGO-based system needs to model uncertainty at all levels: queries, mappings and the underlying data. Of course, all aspects of PAYGO are infused with a pay-as-you-go philosophy.

We are currently building the first PAYGO-based data integration research prototype at Google. We describe some of the challenges in building a PAYGO-based system, and some of our preliminary work in this direction.

The paper is organized as follows. Section 2 details specific cases of structured data on the web, including their scale and challenges. Section 3 outlines the PAYGO data integration architecture. Section 4 describes related work, and Section 5 concludes.

2. STRUCTURED DATA ON THE WEB

In this section, we analyze three scenarios where structured data arise on a web scale and highlight the data integration challenges that arise in each case. First, we use the Deep Web to illustrate the amount of structured data that might lie hidden in web-accessible databases and thereby the potential for structured data techniques on the web. We describe the challenges in making this structured data searchable through current search engines.

Second, we use Google Base to illustrate the extent of heterogeneity that can arise when considering large collections independently-contributed structured data. We describe the possibilities and challenges for structured querying over such heterogeneous data.

As a third example of structured data on the web, we also briefly touch upon annotation schemes (e.g., Google Co-op, Flickr). While such annotations are a much lighter-weight form of semantics, the goal is to add more structure to the web to support better and/or vertical search services (i.e., search within a single domain).

In each scenario, we outline the current techniques used and their limitations. Thus, in addition to exposing the challenges, our goal is to use these scenarios to identify the desiderata for data integration systems of the future that aspire to integrate structured data on a truly web-scale.

2.1 The Deep Web

The deep (or invisible) web refers to content that lies hidden behind queryable HTML forms. These are pages that are dynamically created in response to HTML-form submissions, using structured data that lies in backend databases. This content is considered invisible because search-engine crawlers rely on hyperlinks to discover new content. There are very few links that point to deep-web pages and crawlers do not have the ability to fill out arbitrary HTML forms. The deep web represents a major gap in the coverage of search engines: the deep web is believed to be possibly larger than the current WWW, and typically has very high-quality content [3].

No. of pages with web forms	23.1 million
No. of distinct actions	5.4 million
No. of distinct signatures	3.2 million
No. of web forms with at least one text input	1.5 million
No. of likely deep web sources: at least one text input and between two and ten total inputs	647,000
No. estimated deep web sources (scaled estimate based on an index size of 1B web pages)	25 million

Table 1: Extent of the Deep Web: the number of web forms in a 25-million page random sample of the Google index.

There has been considerable speculation in the database and web communities about the extent of the deep web. We take a short detour to first address this question of extent. In what follows, we provide an estimate of the size of the deep web in terms of the number of forms. This measure will give us an idea of the quantity of structured data on the web and hence the potential and need for structured data techniques on a web-scale.

Extent of the Deep Web: The numbers below are based on a random sample of 25 million web pages from the Google index. Readers should keep in mind that public estimates of index sizes of the main search engines are at over a billion pages, and scale the numbers appropriately.

Our study is summarized in Table 1. To our surprise, we observed that out of 25 million pages, there were 23.1 million pages that had one or more HTML forms. Of course, not all of these pages are deep web sources. Thus, we refined our estimate by attempting to successively eliminate forms that are not likely to be deep web sources.

First of all, many forms refer to the same *action*, i.e., the url that identifies the back-end service that constructs result pages. In our sample, there were 5.4 million distinct actions. Many actions, however, are changed on-the-fly using Javascript and therefore many of them may refer to the same content. As a lower bound for the number of deep web forms, we counted the number of hosts in the different actions urls. We found 1.4 million distinct hosts in our sample.

In order to refine our notion of distinct deep web forms, we computed a signature for each form that consisted of the host in the form action and the names of the visible inputs in the HTML form. We found 3.2 million distinct signatures in our sample.

To further refine our estimate, we decided to count only forms that have at least one text field. After all, if a form contains only drop-down menus, check-boxes and radio buttons, it is conceivable that a search engine can try all combinations of the form and get the content in a domain-independent way. We also eliminated common non-deep web uses of forms such as password entry and mailing list registration. In our sample, 1.5 million distinct forms had at least one text box.

Finally, forms that contain a single input are typically (but certainly not always!) of the type "search this site" and do not yield new web content; similarly, forms with a large number of inputs (say, 10) often capture some detailed interaction, e.g., booking an airplane ticket. To further limit our numbers to forms that are likely to offer deep web content, we counted the number of forms that have at least one text input and between two and ten total inputs. In our sample, we found 647,000 such distinct web forms. This number corresponds to roughly 2.5% of our sample of 25 million pages. Scaling this estimate to an index of one billion pages yields 25 million deep web sources. While a simple scaling of the numbers might not be statistically accurate (as would be true of any *unique* aggregator), the numbers we show will serve to illustrate the scale of the data. We are aware of only one previous study of the number of web forms [8] that estimated 450,000 forms and did not consider any of the filters that led to our final estimate.

To put this estimate in perspective, consider that each deep web source may lead to a large amount of content (e.g., a single form on a used car site leads to hundreds of thousands of pages [6]). Thus, the amount of content on the deep web is potentially huge.

In addition to their large number, we observed that the semantic content of deep web sites varies widely. Many of the sources have information that is geographically specific, such as locators for chain stores, businesses, and local services (e.g., doctors, lawyers, architects, schools, tax offices). There are many sources that provide access to reports with statistics and analysis generated by governmental and non-governmental organizations. Of course, many sources offer product search. However, there is a long tail of sources that offer access to a variety of data, such as art collections, public records, photo galleries, bus schedules, etc. In fact, deep web sites can be found under most categories of the ODP directory [28].

Indexing the Deep Web: Given the estimated number of forms on the web, there clearly lies a potential for the use of structured data techniques. We first consider a more direct question: how do we go about making the content on the deep web searchable through general purpose search engines?

The typical solution promoted by work on web-data integration is based on creating a virtual schema for a particular domain and mappings from the fields of the forms to the attributes of the virtual schema. At query time, a user fills out a form in the domain of interest and the query is reformulated for the relevant forms. In fact, in specific domains (e.g., travel, jobs) this approach is used to create vertical search engines. For example, Transformic Inc., a predecessor to our current efforts, created the web site www.everyclassified.com offering integration of newspaper and online classified sites in six domains. The site was constructed by creating mappings from these sites to carefully designed mediated schemata. Using semi-automated schema mapping techniques (such as those described in [10]), we were able to quickly incorporate over 5,000 different form-based sites in a matter of about two man-months. For general web search, however, the approach has several limitations.

The first limitation is that the number of domains on the web is large, and even precisely defining the boundaries of a domain is often tricky. Hence, it is infeasible to design virtual schemata to provide broad web search on such content. (When such schemata exist, however, we should definitely leverage them).

The second limitation is the amount of information carried in the source descriptions. Although creating the mappings from web-form fields to mediated schema attributes can be done at scale, source descriptions need to be much more detailed, otherwise the number of sites relevant to a query is typically large. With the numbers of queries on a major search engine, it is absolutely critical that we send *only* relevant queries to the deep web sites; otherwise, they will crash. For example, for a car site, it is important to know the geographical locations of the cars it is advertising and the distribution of car makes in its database. Even with this additional knowledge, the engine may impose excessive loads on certain web sites. Further, the virtual approach makes the search engine reliant on the performance of the deep web sources, which typically do not satisfy the latency requirements of a web-search engine.

The third limitation is our reliance on structured queries. Since queries on the web are typically sets of keywords, we need to conduct the reformulation by first identifying the relevant domain(s) of a query and then mapping the keywords in the query to the fields of a mediated schema. This is a hard problem in general and typically requires some knowledge of the contents of the data source and the values that are acceptable in any particular input field.

The above disadvantages led us to consider a *surfacing* approach to deep web content. In this approach, deep web content is surfaced by simulating form submissions, retrieving answer pages, and putting them into the web index. The main advantage of the surfacing approach is the ability to re-use existing indexing technology; no additional indexing structures are necessary. Further, a search is not dependent on the run-time characteristics of the underlying sources because the form submissions can be simulated off-line and fetched by a crawler over time. A deep web source is accessed only when a user selects a web page that can be crawled from that source, and then the user gets the most up-to-date content. Similarly, the problem of identifying relevant deep-web sites is mostly avoided because web search is performed on HTML pages as before. In terms of schemata and source descriptions, the needs of the surfacing approach are lighter. Instead of creating schemata for individual domains and providing detailed source descriptions, it is enough to identify some way of crawling the data by filling in values for a subset of the form fields (in a sense, finding some access path to the data). Hence, this approach can be more easily applied to a much broader collection of domains.

While surfacing does make deep web content searchable, it has some important disadvantages. The most significant one is that we lose the semantics associated with the pages we are surfacing by ultimately putting HTML pages into the web index. By doing so, we overlook the possibility of exploiting the structure during query time. Further, it is not always possible to enumerate the data values that make sense for a particular form, and it is easy to create too many form submissions that are not relevant to a particular source. For example, trying all possible car models and zip codes at a used-car site can create about 32 million form submissions – a number larger than the number of cars for sale in the United States. Finally, not all deep web sources can be surfaced: sites with robots.txt as well as forms that use the POST method cannot be surfaced.

We would ideally like a solution where given an arbitrary user keyword query, we identify just the right sources that are likely to have relevant results, reformulate the query into a structured query over the relevant sources, retrieve the results and present them to the user. The problem of identifying relevant sources to user keyword queries, which we call *query routing*, will be a key to any web-scale data integration solution as we discuss later. We are currently pursuing the surfacing approach as a first step to exploring the solution space in the context of the deep web.

2.2 Google Base

The second source of structured data on the web that we profile, Google Base, displays a high degree of heterogeneity. Here, we detail this degree of heterogeneity and describe some of the challenges it poses to web-scale data integration.

Google Base is a recent offering from Google that lets users upload structured data into Google. The intention of Google Base is that data can be about *anything*. In addition to the mundane (but popular) product data, it also contains data concerning matters such as clinical trials, event announcements, exotic car parts, and people profiles. Google indexes this data and supports simple but structured queries over it.

Users describe the data they upload into Google Base using an *item type* and attribute/value pairs. For example, a classified advertisement for a used Honda Civic has the item type vehicle and attributes such as make = "Honda" and model = "Civic". While Google Base recommends popular item types and attribute names, users are free to invent their own. Users, in fact, do often invent their own item types and labels, leading to a very heterogeneous collection of data. The result is that Google Base is a *very large*, *self-describing, semi-structured, heterogeneous database*. It is self-describing because each item has a corresponding schema (item type and attribute names). It is semi-structured and heterogeneous because of the lack of restrictions on names and values.

Content Heterogeneity: Heterogeneity in Google Base occurs at the level of item types, attributes, and attribute values.

Less than a year since its launch, Google Base already contains over 10,000 item types that together contribute almost 100,000 unique schemata (a unique set of attribute names associated with some item). There are over 1000 item types that have more than 10 items. While there are popular item types such as products, reviews, vehicles, and housing, there are also a large number of more esoteric item types like high performance car parts and marine engine parts. In addition to the sheer complexity of handling such a number of item types, we also see two new challenges to schema management relating to Google Base. First, the large number of item types form a specialization hierarchy. For example, an item can alternately be described as a product, a car part, or a high performance car part. Users choose such different item types naturally and this leads to a proliferation of item types. Second, in addition to naturally occurring heterogeneity, we find that heterogeneity occurs in a different, almost malicious way. Users provide different item-type names or attribute names in an attempt to improve their ranking. Hence, we are witnessing a kind of *database design by the masses*. Returning to our main themes, the result of these challenges is that we need a very different approach to schema management in the context of the web to understand this high degree of heterogeneity.

To get a feel for the attribute-level heterogeneity, we looked at the items of type vehicle, and found that there are over 50 distinct attribute names that occur in 10 or more items. While this might seem to be a large number, there is a core set of attributes that appear in a large number of items. This includes common attributes like make, model, location, color, and price. A commonly occurring type of attribute-level heterogeneity is one of complex attributes. For example, color for some items includes both the internal color and the external color of a vehicle, while for others they are separate attributes.

For heterogeneity at the level of attribute values, we considered the different values for the attribute color of vehicles, and found that there are over 250 different colors with 5 or more items. In addition to more frequent colors like "silver", "white", and "black", there are cars with the colors "polished pewter" and "light almond pearl metallic". An interesting heterogeneity challenge arises in the extrapolation of domain specific similarities for attribute values (e.g., "polished pewter" is similar to "metallic silver") and incorporating these similarities into query processing. We note that the presence of structure (all the values mentioned in this case are colors of cars) makes it possible to consider performing such reconciliation, e.g., using domain-specific similarity measures. Such reconciliation would not be possible for purely unstructured data.

Querying Google Base: Given that its contents are structured, in principle it is expected that Google Base support semantically-rich querying. However, web search users typically prefer keyword queries and in most cases do not possess the expertise to pose well-defined structured queries. As a result, Google Base faces two main challenges: query routing to determine relevant item types and query refinement to interactively construct well-specified structured queries. We illustrate these challenges by describing three modes of querying Google Base supports, in increasing levels of difficulty.

In the first mode of querying, the user specifies a particular item type and perhaps provides values for some of the attributes (but not all). This is essentially a structured query, but Google Base tries to help the user refine it further by proposing a set of attributes with drop-down lists of candidate values. For example, choosing vehicle as an item type leads to possible refinements based on the attributes make, model, color, and location. The user can further continue to restrict search results by choosing specific values in the drop-downs for the attributes. With each successive user refinement, Google Base recomputes the set of candidate refinements.

Google Base proposes query refinements by computing histograms on attributes and their values during query time. It chooses the attributes that best help the user refine the current query. For example, if the user is interested in "Honda Civic" and the location is restricted to "Mountain View, CA", then the drop-down values for color change to reflect only the colors of Honda Civics available in the Mountain View area.

In the second mode of querying, the user poses a keyword query over *all* of Google Base. This method is the main mode of querying on base.google.com. Here, Google Base must first suggest a set of item types to show the user and perhaps a few example answers. Hence, Google Base faces the problem of query routing between different item types. For example, for the query "Honda Civic", Google Base proposes the item types vehicle and review. By choosing one of the suggested item types, the user restricts the search results to items of the chosen type.

The third mode of querying, and by far the most common one, is a keyword query on the main search engine, google.com. Here the user may not even be aware of the existence of Google Base and is looking for any answer that may be relevant. In addition to a very challenging query-routing problem (i.e., determining if the engine should send a query to Google Base), there is the problem of ranking answers across properties. For instance, the query "Honda Civic Mountain View, CA" may return answers from multiple item types in Google Base, from listings of Honda dealers in the bay area (stored in the Google Local database), and from web documents that may be highly ranked w.r.t. the query.

By iteratively proposing well-chosen query refinements, Google Base enables users to construct structured queries. The better defined queries as well as the ability to use domain-specific similarity measures and ranking criteria lead to an enhancement in search experience.

While the above discussion demonstrates the ability to exploit structure, observe that Google Base only queries over structured data that it has complete knowledge about. In the case in a more general web data integration solution, however, such knowledge does not exist. Such a solution will also have to incorporate sources with only source descriptions and summarized data contents. Such a setting will further exasperate the heterogeneity challenges that are in evidence in Google Base.

2.3 Annotation Schemes

A growing fraction of structured data on the web is the content created by a variety of *annotation schemes*. Annotation schemes enable users to add tags describing underlying content (e.g., photos, products, reviews, urls) to enable better search over the content. The Flickr Service by Yahoo! is a prime example of an annotation service for photo collections. von Ahn [33] took this idea to the next level by showing how mass collaboration can be used to create high-quality annotations of photos.

Recently, Google created the Google Co-op service that enables users to create customized search engines (see http://data.cs.washington.edu/coop/dbresearch/index.html for a search engine developed for the database research community). To create a custom search engine, a user identifies a set of labels or categories (known as *facets* in Google Co-op) that are of interest to the community at hand. For example, in the case of database research, these facets include PROFESSOR, PROJECT, PUBLICATION, JOBS. Web pages that fit the various categories are manually (or otherwise) annotated, e.g., we would annotate the homepage of a database faculty with facet PROFESSOR and pages that appear in DBLP with the facet PUBLICATION. These annotations are uploaded in an XML format to Google Co-op. The annotations can be used to either query for web-pages that specify a particular facet, or to refine the results returned for an earlier query. Facets can also be automatically triggered, e.g., if we were to search for data integration publications, the PUBLICATION facet will be triggered, and the answers would include only the pages annotated as publications.

The important point to note is that the annotations are very lightweight – we only annotate to which facet the page belongs and nothing else. Further, the integration problems faced in the context of annotation schemes are somewhat simpler. Typically, the annotations can be used to improve recall and ranking for resources that otherwise have very little meta-data associated with them (e.g., photos, videos). In the case of Google Co-op, customized search engines can specify query patterns that trigger specific facets as well as provide hints for re-ranking search results. The annotations that any customized search engine specifies are visible only within the context of that search engine. However, as we start seeing more custom search engines, it would be desirable to point users to different engines that might be relevant.

The structure of data contributed by annotation schemes differs in granularity from that in Google Base (simple aboutness tags as opposed to attribute-value pairs) and the deep web. In fact, there is a broad spectrum of structured data on the web. We highlight this aspect of web data in [27]. Any web-scale data integration system must gracefully incorporate structured data of varying granularity.

2.4 The Web-scale Data Integration Challenge

As observed in this section, there is a huge amount of structured data already on the web, and it is going to increase further in the coming years. This clearly presents the need and an opportunity for query answering techniques that make use of the structure. The utility of structure-aware search engines has already been demonstrated to some extent by the success of specialized search engines in specific domains (e.g., travel, weather and local services). Handling content in a domain-specific way can potentially lead to better ranking and refinement of search results.

But, by its very nature, the structured data will be very heterogeneous and current data integration architectures cannot cope with this web-scale heterogeneity. The crux of the problem lies in the fact that data on the Web is about *everything*. If the domain of structured data on the web were well defined, many of the issues we outlined above would disappear (as in the case of successful specialized search engines).

One could argue that the problem is simply that there are many domains (perhaps a few hundred) and we should simply model them one by one. The issue, however, is much deeper. Data on the Web encompasses much of human knowledge, and therefore it is not even feasible to model all the possible domains. Furthermore, it is not even clear what constitutes a single domain. Facets of human knowledge and information about the world are related in very complex ways, making it nearly impossible to decide where one domain ends and another begins. The only broad attempt to model human knowledge, the Cyc Project [24], has been going on for about two decades and has met with only limited success.

Even if it were possible to build a representation of all the domains that appear on the Web, the representation would be inherently brittle and hard to maintain. It is hard enough to manage heterogeneity in a single domain, imagine trying to do it at such scale. And if that was not bad enough, the representation needs to be maintained in over a hundred languages.

Traditional Data Integration	PAYGO **Data Integration**
Mediated Schema	Schema clusters
Schema mappings	Approximate mappings
Structured queries	Keyword queries with query routing
Query answering	Heterogeneous result ranking

Table 2: Comparison of components in traditional data integration and the PAYGO architecture.

To offer data integration services at web scale, we need a system that can model arbitrary relationships in the world, can evolve over time and offer best-effort answers at any point. The PAYGO-integration architecture we describe in the next section attempts to address challenges.

3. THE PAYGO ARCHITECTURE

Having described specific challenges facing an increasingly structured web, we now outline the PAYGO data integration architecture designed to address these challenges. We first discuss the architectural components and principles underlying PAYGO contrasted with traditional data integration architectures and then describe these principles in more detail in the form of a research prototype we are building at Google based on the PAYGO architecture.

PAYGO **Components and Principles:** We begin by briefly recalling the main components of a traditional data integration system, and then contrast them with the components of PAYGO.

A traditional data integration system offers uniform access to a set of data sources through a *mediated schema*. The mediated schema is purely logical; the data itself remains in the data sources. To relate the contents of the data sources with the elements of the mediated schema, a data integration system uses a set of *semantic mappings*. Various languages have been proposed for specifying these mappings [20, 25] and multiple techniques developed for creating these mappings semi-automatically [30].

A query to a data integration system is specified in some subset of SQL or XQuery, and is formulated in terms of the mediated schema. When a query is posed, it is *reformulated* to access the data sources. The reformulation process uses the semantic mappings to (1) determine which data sources are relevant to the query, (2) formulate appropriate sub-queries on the individual data sources, and (3) specify how results from the different sources are combined (e.g., via join or union) to produce answers to the query.

In the context of data integration on the web, however, these techniques are no longer adequate. Thus, we propose a new data integration architecture, PAYGO, that evolves the components described above as well as incorporates new techniques to handle the scale and heterogeneity of structured web data. See Table 2 for a side-by-side comparison of traditional data integration components with those in PAYGO.

Schema clustering: First, in PAYGO we cannot rely on a single mediated schema. As described earlier, modeling data at web scale requires that we model *everything*. Hence, instead of a mediated schema, in PAYGO we have a repository of schemata. Schemata in the repository will be *clustered* by topic. In addition, some schemata may be treated in a special way. For example, we may have selected schemata for particular domains, and these schemata will act as hubs to which we strive to map other sources when possible. Note that some schemata may belong to multiple clusters (e.g., a data source about culture events in San Francisco may belong to a clusters about culture, travel, events and artists).

Approximate schema mapping: In the presence of heterogeneity at web scale, we cannot assume that exact semantic mappings between data sources exist. Not only is it hard to create such mappings and maintain them at web scale, but given the extremely broad domain range of the data we must handle, it is not even clear if there always exists a single correct mapping. Thus, schema mappings in PAYGO are fundamentally approximate. At the most extreme case, a schema mapping can simply be a statement that two schemata are related to each other and belong to the same cluster. In other cases, the schema mapping may be the output of an automated schema mapping module, and hence be partially uncertain. At the other end of the spectrum, a human will inspect a mapping and correct it if necessary, thus deeming it certain. Unlike a traditional data integration system where schemata are mapped to a single mediated schema, in PAYGO schema mappings can exist between any pair of sources. Therefore, a query in PAYGO can be reformulated over many intermediate and data source schemata thus allowing for greater semantic integration. This is similar in spirit to the multiple reformulations as in a peer data management system (PDMS) [21].

Keyword queries with routing: The vast majority of queries on the web are keyword queries; thus, we assume that queries to a PAYGO-based system will be posed as such. Of course, in cases where the user-interaction can support query refinement, we may solicit help from the user to structure the query. Hence, as a first step, we take a keyword query and try to reformulate it as a structured query. These reformulations, of course, produce necessarily uncertain queries. We then proceed to select the relevant data sources, but we need to do so in a different way since our mappings are approximate and schemata of sources are organized in clusters. We term this entire process *query routing*.

Heterogeneous result ranking: Finally, due to the approximate nature of mappings and keywords in PAYGO, there no single true answer to a given query. Instead, query answering in PAYGO involves *ranking*. Ranking is complicated by the fact that our query and mappings are approximate to start with, and that we handle multiple heterogeneous properties (e.g., we need to compare answers from deep web sources, Google Base and unstructured sources).

Before we discuss the components of PAYGO in more detail, we emphasize two over-arching principles underlying the architecture.

Pay-as-you-go integration: At web-scale, integration is an ongoing process. The process starts with disparate data sources, and incrementally improves the semantic glue amongst them. Thus, over time, the data sources are increasingly integrated thereby improving the ability to share data between them. However, at any point during the ongoing integration, queries should be answered as best possible using the available and inferred semantic relationships. This approach is unlike a traditional data integration system that requires more complete knowledge of semantic relationships. In PAYGO, we emphasize a set of mechanisms that enable us to improve the semantic relationships over time. The mechanisms are a combination of automated techniques that find and/or suggest relationships, and of techniques that involve feedback from users about such guesses. As we describe the components below, we highlight the opportunities for employing these mechanisms.

Modeling uncertainty at all levels: As is clear from the discussion above, a system based on the PAYGO architecture needs to model uncertainty at multiple levels: data, semantic mappings and queries. It is desirable that PAYGO have a principled and uniform formalism for modeling uncertainty at all these levels, so uncertain decisions made at one level can propagate appropriately to the other levels. Developing such a formalism for modeling uncertainty is a subject of ongoing work, and involves balancing multiple competing desiderata: generality and expressive power, ease of use and understandability, and the ability to process queries and data efficiently. Hence, our current focus is first on clearly understanding the requirements for such a formalism.

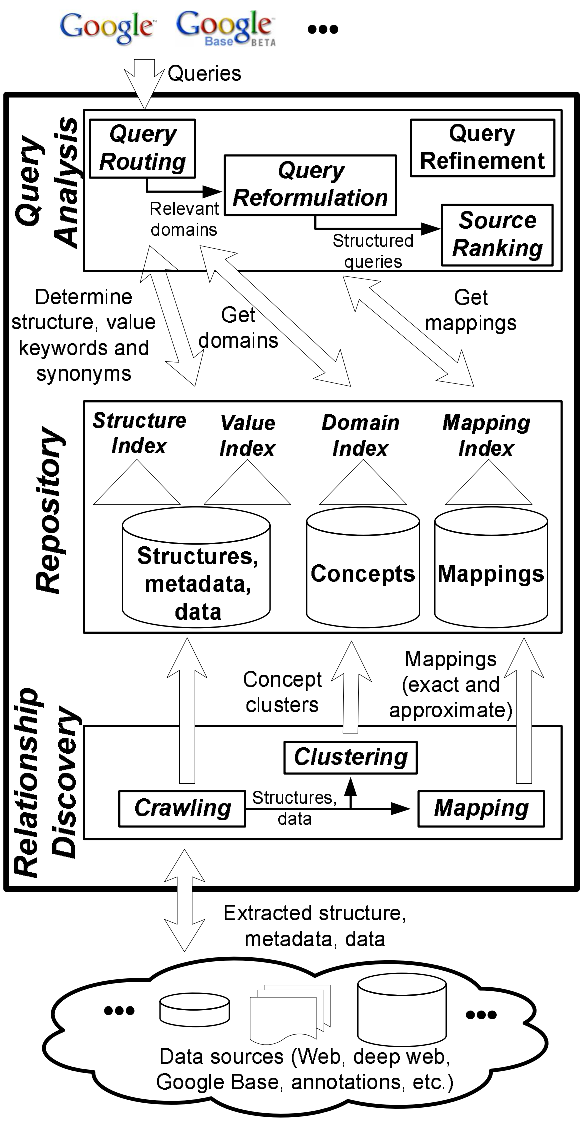

Figure 1: An instantiation of the PAYGO data integration architecture.

3.1 A PAYGO-based Data Integration System

Based on the architecture described above, we have begun to build a research prototype PAYGO-based data integration system at Google for managing web-scale heterogeneity. Here, we briefly describe the elements of this system and some of our initial research directions. Figure 1 shows the different components of the system.

The metadata repository: The metadata repository stores the schemata and the mappings currently known to the system, and provides a query interface over this collection for use by the other components of the system. As described earlier, while we do not have a single mediated schema for the system, some schemata in the repository may be treated in a special way. For example, we may choose a particular schema for the car domain to which we would like to map other schemata when possible. Such schemata will be used to focus the the schema mapping algorithms.

The repository addresses several challenges. First, it stores schemata that exhibit a high degree of variance in their internal structure. For example, annotation schemata have very little structure compared to Google Base schemata, and these differ from web-form schemata. Second, it supports a variety of query interfaces, such as (1) find all schemata in which model appears in some attribute name, and (2) *nearest schema queries*, i.e., find a schema closest to a given one. Finally, the repository tracks the lineage of the mappings in the system such that when a mapping is changed, other mappings that depend on it are reconsidered.

Schema clustering and mapping: Relationships between schemata can be specified in terms of semantic mappings, clustering, or both.

For schema clustering, we represent each schema as a set of feature vectors and use both the structure and data information from the data sources for determining the clusters. Each vector extracts some features from the schema or the data represented by the schema. We design customized scoring mechanisms to compare the feature vectors and therefore determine the distance between two schemata. The set of major features we consider and their comparison mechanisms we adopt include the following: the schema's attribute names vector (compared using linguistic similarities), the data values vector (compared using linguistics similarities or data type matching), and finally sets of commonly occurring keyword groups. Based on these features, we establish a large graph of schemata with edge weights denoting the distance between schemata. We then create clusters by detecting groups of schemata that are closely connected in the graph.

For schema mapping, we employ the Corpus-based schema matching technique [26] to leverage the large number of schemata we have to produce mappings. Of course, mappings resulting from this and other automated schema matching systems are approximate by nature. In some cases, the mappings may be important enough that we let a human inspect it and verify it. However, given the scale of the number of sources, we will not be able to do that in the vast majority of the cases.

In [11] we describe semantics of approximate schema mappings. We identify two semantics for approximate mappings. The first, called *by-table semantics*, assumes that there is a single mapping that correctly describes the relationship between a pair of schemata, but we are uncertain which one it is. In the second, called *by-tuple semantics*, the correct mapping may depend on the particular tuple in the source. We also describe the complexity of answering queries in the presence of approximate mappings, and consider several methods for concisely representing approximate mappings.

Query reformulation and answering: Queries in a PAYGO-based system are posed as keywords. The query answering component begins by generating a structured query from the given query, routing it to the relevant sources, and then ranking the answers coming from the multiple sources. We now describe a preliminary algorithm for answering queries in a PAYGO-based system. In particular, the algorithm illustrates how the novel components of the architecture interact with each other. We use the example query "Honda Civic Review" in our description.

Step 1. Classify keywords: We first divide the keywords in the query into those corresponding to data values and those corresponding to attribute names (or other schema elements). For example, "Honda" and "Civic" are values of make and model, respectively, and "Review"' is an attribute name. Of course, many keywords can play multiple roles. They can be values in multiple domains (e.g., the classic "Jaguar" example), or even serve both as data values and attribute names. To make this classification, we look up the keywords in two indexes provided by the repository: the *structure index*, that indexes schema elements of data sources, and the *value index*, that indexes data values.

Step 2. Choose domain: We use the schema clusters to determine which domains the user query most likely corresponds to. This step proceeds by a lookup into the *domain index*, that indexes the structure terms associated with each cluster (i.e., domain). Note that one structure term can belong to several different domains. In our example, we find out that review, make and model all belong to the vehicle domain (among others). We keep only the domains that are deemed relevant to the query.

Step 3. Generate structured queries: We generate candidate structured queries for each relevant domain by trying to match the keywords in the query to common attribute names in that domain. This step utilizes our algorithms for answering queries using approximate schema-mapping. For example, "Honda Civic Review" is transformed into "make, model, review" for the vehicle domain. Here, too, there will possibly be many structured queries, so we record each one with a degree of certainty of the mapping.

Step 4. Rank sources: We select a set of relevant sources and rank them. We use the structure-index to find the sources that have terms used in the structured queries resulting from Step 3. We rank the answers based on factors including (1) how many required structure terms are involved in the data source (so the data source containing "make, model, review" is ranked higher than the one containing only "make, model"), (2) whether the involved terms match the original query better, and (3) the results of previous search experiences, namely, what sources users went to when they posed similar queries.

Step 5. Heterogeneous Result Ranking: Of course, with potentially large numbers of results, the system must rank the results before returning them to the user. The challenge of ranking in this context, however, is that the results come from a large number of diverse sources with varying degrees of structure, not just text documents as in standard IR or even data from similar data sources such as in work combining databases with IR [2].

Thus, result ranking in this context involves the traditional IR ranking challenges compounded by the diversity of the data sources and the uncertainty in structural information in both the schemata and query. Specifically, the result ranking mechanism takes into account the following factors:

- Traditional IR metrics for the quality of the data item itself based on relevance, past queries [1], and other factors
- The uncertainty in the the query reformulation
- The uncertainty of the schema on which the query was based (i.e., schema extraction problems)
- The uncertainty of the schema mapping on which the query's reformulation was based

3.2 Pay-as-you-go in PAYGO

As discussed earlier, one of the main premises of the PAYGO architecture is that it is impossible to fully integrate all sources beforehand. The pay-as-you-go principle states that the system needs to be able to incrementally evolve its *understanding* of the data it encompasses as it runs. By understanding, we mean knowledge of the underlying data's structure, semantics, and relationships between sources.

We employ several automated methods to bootstrap the understanding of our underlying data. Some of these have already been described earlier, including automated schema mapping and schema clustering. In addition, we employ techniques for discovering additional relationships between data in different sources (in the spirit of Semex [12], iMeMex [5], and DBLife [9]), techniques for reconciling references to the same real-world objects (e.g., [13]), and techniques for information extraction from text.

Ideally, the results of all these automated methods should be verified by humans, who can quickly correct the errors. However, at web scale, it is impossible for humans to inspect all of these results. Hence, it is crucial that we leverage as much as possible feedback we can get from users or administrators of the system [19].

In many cases, we can leverage implicit feedback from users. The prime example of implicit feedback is to record which of the answers presented to the user are clicked on. Such measurements, in aggregate, offer significant feedback to our source selection and ranking algorithm. Observing which sources are viewed in conjunction with each other also offers feedback on whether sources are related or not. Finally, when the user selects a refinement of a given query, we obtain feedback on our query structuring mechanisms.

In other cases, we may be able to solicit explicit help from the user by asking her questions regarding the system's understanding of the underlying data (e.g., "is this mapping correct?"). At web-scale, however, there are far too many possible questions to ask the user; thus, the challenge is to determine *which* questions to ask. This challenge is compounded by the fact that multiple different types of questions, generated by different mechanisms, must be compared (e.g., schema mapping questions versus entity resolution questions).

Our approach solving this problem is to employ *decision theory* [32] to determine which questions to ask the user in a principled manner. The first step in translating this problem into decision theoretic terms is to quantify this benefit, or *utility*, of each question in terms that apply across multiple types of questions produced by different mechanisms. The goal of this system is to provide better (or, ideally, perfect) answers to user's queries; thus, we define a *utility function* that captures user satisfaction on queries. This function is a numerical value that quantifies answer quality such as precision/recall, F1 measure, or average precision [31].

However, since the outcome of a question cannot be known in advance, the system has to take into account the uncertainty associated with each question. For instance, if we are very uncertain of a given mapping, then our *expected* gain in utility will be small for asking this question as it is likely we will gain very little information.

Decision theory formalizes this process through the calculation of the *value of perfect information* (VPI). Basically, the VPI score for a given question states what is the expected benefit of knowing the correct answer to the question given the uncertainty in the outcome of the question and the potential benefit of each of the possible outcomes. We are currently developing a decision theoretic-based framework that utilizes these formalisms to provide the system with a ranked set of questions to ask the user(s).

The final pay-as-you-go-based technique we are incorporating in our system is immediate feedback through data visualization. Users are very fickle and, in general, don't like doing work; we believe that when users can immediately see the results of their efforts in structuring data, they are more enticed to invest these efforts. For example, consider the process of creating a mash-up of two data sources. In order for the mash-up to display properly, it is imperative the the columns being joined have data values drawn from the same domain. Hence, at this step the user must make sure that references are reconciled across the two sources.

4. RELATED WORK

There has been a lot of recent interest in web-data integration systems. The work closest to ours is the Meta-Querier project [7] that investigates the integration of deep-web sources. Their approach focuses on automatically creating a unified interface for accessing multiple deep-web sources in specific domains, i.e., a vertical search site. Our goal is more general – we do not want to create vertical search sites in specific domains, but rather enhance web-search by including content in deep web and other structured data sources across many domains. The schema matching component in PAYGO builds on ideas proposed in earlier matching works such as [10, 22, 23, 26, 34]. Our goal is to provide the matching component as one of the services in our dataspace platform.

Our metadata repository manages large collections of schemata. While similar in spirit to model management [4], our repository stores approximate mappings and schema clusters rather than exact mappings. Our repository is designed to store a much larger number of schemata (at least in the thousands), but the schemata in general are much simpler. Further, we do not consider the wide range of model management operations described in that work.

While there has been much speculation on the extent of the deep web, our estimate of the number of web forms is the first empirical study based on a crawl on the WWW. Earlier such estimates were either based on a manual study [3] or by polling web servers [8]. We describe two alternatives for enhancing web search over deep-web content. While much of the web-integration works have focused on virtual integration, to the best of our knowledge, only one earlier work [29] considered surfacing, though in a limited context, as a means of crawling deep-web content.

5. CONCLUSIONS

There is a prevailing sense in the database community that we missed the boat with the WWW. The over-arching message of this paper is that a second boat is here, with staggering volumes of structured data, and this boat should be ours. To back this up, we offered several statistics on the current trends in web content, and outlined some of the major challenges in searching it.

From the perspective of data management, we are used to thinking in terms of a dichotomy between structured data versus unstructured data, and the Structure Chasm that lies between them [18]. Web-scale heterogeneity, i.e., data in millions of disparate schemata, presents a unique point between the two ends of the chasm. The goal of the PAYGO architecture described in this paper is to show that data management techniques can be applied to these collections of data, but to do so, we need to change some of the assumptions we typically have when dealing with heterogeneous data.

Finally, while we are fortunate at Google to have access to incredible amounts of data and queries, an important component of our plan is to make public certain data sets that will enable the community at large to make progress on these important challenges.

6. REFERENCES

[1] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *cidr*, 2003.

[2] S. Amer-Yahia. Report on the DB/IR Panel at SIGMOD 2005. 34(4), December 2005.

[3] M. K. Bergman. The Deep Web: Surfacing Hidden Value, 2001. http://www.brightplanet.com/technology/deepweb.asp.

[4] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[5] L. Blunschi, J.-P. Dittrich, O. R. Girard, S. K. Karakashian, and M. A. V. Salles. A dataspace odyssey: The iMeMex personal dataspace management system. In *CIDR*, 2007.

[6] Cars.com FAQ. http://siy.cars.com/siy/qsg/faqGeneralInfo.jsp#howmanyads.

[7] K. Chang, B. He, and Z. Zhang. toward large scale integration: Building a MetaQuerier over databases on the web. In *CIDR*, 2005.

[8] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, 33(3), 2004.

[9] P. DeRose, W. Shen, F. Chen, Y. Lee, D. Burdick, A. Doan, and R. Ramakrishnan. DBLife: A community information management platform for the database research community. In *CIDR*, 2007.

[10] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

[11] L. Dong, A. Halevy, and C. Yu. Approximate schema mappings. Submitted for publication, 2006.

[12] X. Dong and A. Halevy. A Platform for Personal Information Management and Integration. In *CIDR*, 2005.

[13] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Sigmod*, 2005.

[14] Flickr. http://www.flickr.com.

[15] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.

[16] Google co-op. http://www.google.com/coop.

[17] Google Base. http://base.google.com.

[18] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[19] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, 2006.

[20] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[21] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.

[22] B. He and K. C.-C. Chang. Statistical schema integration across the deep web. In *Proc. of SIGMOD*, 2003.

[23] H. He, W. Meng, C.T.Yu, and Z.Wu. Wise-integrator: an automatic integrator of web search interfaces for e-commerce. In *VLDB*, 2003.

[24] D. B. Lenat and R. Guha. *Building Large Knowledge Bases*. Addison Wesley, Reading Mass., 1990.

[25] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.

[26] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.

[27] J. Madhavan, A. Halevy, S. Cohen, X. Dong, S. R. Jeffery, D. Ko, and C. Yu. Structured Data Meets the Web: A Few Observations. *IEEE Data Engineering Bulletin*, 29(4), December 2006.

[28] ODP – Open Directory Project. http://dmoz.org.

[29] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, 2001.

[30] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[31] C. J. V. Rijsbergen. *Information Retrieval*. Butterwoth-Heinmann, 2nd edition edition, 1979.

[32] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[33] L. von Ahn and L. Dabbish. Labeling Images with a Computer Game. In *ACM CHI*, 2004.

[34] W. Wu, C.T.Yu, A. Doan, and W.Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Sigmod*, 2004.