Bulletin of the Technical Committee on

Data Engineering

December 2006 Vol. 31 No. 4 **IEEE Computer Society**

Letters

Special Issue on Special Issue on Web-Scale Data, Systems, and Semantics

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

There are two Data Engineering Bulletin web sites: http://www.research.microsoft.com/research/db/debull and http://sites.computer.org/debull/.
The TC on Data Engineering web page is http://www.ipsi.fraunhofer.de/tcde/.

Letter from the Editor-in-Chief

Election Results

The Technical Committee on Data Engineering (**TCDE**) is the sponsoring technical committee within the IEEE Computer Society both for the Data Engineering Bulletin that you are currently reading and for the International Conference on Data Engineering (ICDE). The members of the TCDE elect a chair who then appoints an executive committee. Paul Larson of Microsoft, who was put forward by the nominating committee, has been elected as the new chair in a just completed election. Erich Neuhold of Fraunhofer is the retiring chair of the TCDE.

The Data Engineering Conference (**ICDE**) is the flagship conference of the TC on Data Engineering. Erich Neuhold is also the retiring chair of the ICDE Steering Committee, the committee that organizes and oversees ICDE. During Erich's tenure, ICDE has become one of the most highly selective, top rated conferences in the database area. His replacement, elected this fall by steering committee members, is Calton Pu of Georgia Tech.

The Current Issue

”Web-scale” really does mean something new. I can personally attest to this, having been involved in a design effort for a web-scale system. Web-scale platforms give a new meaning to system size– think hundreds or thousands of processors, hundreds or thousands of disks, ultra-high availability, rock bottom hardware, software and operations costs... These characteristics force a re-thinking of almost everything about database and associated systems. For example, low cost suggests using cheap disks. High availability perhaps suggests the opposite. To achieve both simultaneously suggests resorting to pervasive replication using the cheap disks. Driving down costs implies automating operations since people are increasingly the high cost component. Even power costs become important, something we have rarely considered in the past.

I was delighted when Dan Suciu, our current issue editor, suggested web-scale as the topic for the December issue. The issue contains papers from a cross section of the ”web scale” industry, Yahoo, Google, IBM, and Microsoft. Getting some clue about what these companies are doing at web scale should be of intense interest our entire database community. The issue contains, as well, two interesting articles about querying web information. ”Web scale” represents a very important part of the future of the database field. You can surely expect web scale to be a relevant research topic for many years to come. I want to thank Dan for a fine job in suggesting and bringing this issue together. I can strongly recommend it to you.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

The field of data management faces new challenges generated by Web-scale data. The amount and kind of data found on the Web, and the need to offer large-scale Web services for email, IM, social networking, photo sharing, etc, requires us to rethink the data models, the processing models, and the platform architectures for supporting large-scale services. Increasingly, more Web "documents" have in fact structure, and are best modeled using database concepts, such as schemas, attributes, and tuples. Web communities generate huge amounts of both structured and unstructured data, such as user profiles, blogs, listings, advertisements. Large-scale, high-availability Web services require a re-design of the data management infrastructure and find new ways to operate it. In this issue of the Data Engineering Bulletin we have invited six papers to illustrate the new challenges faced by Web-scale data. Most of this research is done at the major Web portal companies, which have access to the data, have an infrastructure for search, offer large-scale Web-services, and have the financial resources needed to build the expensive platforms that can cope with Web-scale. But new ideas also emerge from research groups in academia, who are already thinking about the next generation Web-scale data management techniques.

We start with a paper by Bernstein, et al from Microsoft Research that describes very clearly the challenges and tradeoffs faced by large Web services platforms. The authors report from their own experience with Windows Live^{TM}, and discuss design tradeoffs in such large scale platforms. The second paper, from Yahoo! Research, illustrates a different set of issues that arise from managing and combining Web communities, Web services, and advertising services, and highlights the need to support both high-throughput storage/retrieval and analytical processing.

In the next paper Madhavan et al. from Google discuss the structure of data on the Web. It includes a classification of the types of structure, and a discussion on querying both structured and unstructured data. Alba et al. from IBM Research, present in the fourth paper their experience in designing and building the IBM Semantic Super Computing platform, which stores, manages and queries billions of documents.

The last two papers of this issue are representative of the new ideas coming from academic research groups. Agichtein, from Emory University, is proposing to understand the user needs, in order to better guide the information extraction tasks, while Cafarella et al. from the University of Washington discuss novel approaches for expressing queries over both structured and unstructured data on the Web.

Dan Suciu
University of Washington

Data Management Issues in Supporting Large-Scale Web Services

Philip A. Bernstein, Nishant Dani, Badriddine Khessib, Ramesh Manne, David Shutt
Microsoft Corporation
Redmond, WA, 98052-6399
{phil.bernstein, nishant.dani, badriddine.khessib, ramesh.manne, david.shutt}@microsoft.com

Abstract

This paper discusses technical problems that arise in supporting large-scale 24$\times$7 web services based on experience at MSN with Windows Live^{TM} services. Issues covered include multi-tier architecture, costs of commodity vs. premium servers, managing replicas, managing sessions, use of materialized views, and controlling checkpointing. We finish with some possible research directions.

1 Introduction

Large-scale web services supported by fee-for-service or advertising revenue are heavy users of database technology. Such services include email, instant messaging, social networking, blogs, photo sharing, maps, shopping, and classifieds. They also include search services, though these typically use custom indexing engines, not relational database systems.

By large-scale, we mean systems with millions of users and thousands of servers. For example, Windows Live Messenger has thousands of servers and tens of millions of simultaneous online connections. It processes hundreds of millions of logins per day and has over 250 million unique users per month. Each login translates into queries to backend database servers for authentication and authorization. Users send billions of messages per day. The message transfer itself takes place independently of the database. Similarly large numbers have been reported for Hotmail® [1, 2].

These types of services are big business. The trailing 12-month revenue of Google, Windows Live, and Yahoo! combined is over \$16B/year, most of which comes from such services. This annual revenue is comparable to the size of the market for database software products.

In general, these services require a lot of computing power, which is expensive to acquire and operate. A high-end commodity server to run a database system application currently costs upwards of \$20K. A typical such system has 1-4 processors, 2-24GB of main memory, and 12-24 SCSI (Small Computer System Interface) disk drives. The size and technology of the disk system is a strong determinant of overall system price. For example, in August 2006 the best price/performance reported on TPC-C uses a Dell server with one Intel dual-core processor, 24GB of RAM, and 90 SAS (serial-attached SCSI) disk drives [3]. The disk subsystem cost is approximately \$45K out of a total system cost of \$64.5K. If a RAID configuration is needed for high availability, then the costs rise even higher. Multiplying that by the number of servers required to run a large-scale web service, we see that the system can cost tens or hundreds of millions of dollars.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In addition to hardware cost, there is the cost of a building to house the system, power to run it, and system managers to keep it running. Minimizing these costs is critical to a company's ability to offer the service at competitive fees-for-service or advertising rates and still earn a profit. In this paper, we explore the sources of these costs and optimizations to reduce them.

2 System Architecture

To enable scalability across a large number of machines, most back-end systems supporting a large-scale web service at Microsoft use a fairly classical multi-tier design. Users submit transactions, which can be queries or updates, usually via a web browser. Each transaction request is received by a web server, which routes the request to a mid-tier server, which in turn calls database servers. In some cases, such as Windows Live Messenger, a client process communicates using SOAP directly to a mid-tier server with no web server in between.

Usually, the web servers, mid-tier servers and database servers run on different machines. Machines running web servers and mid-tier servers have limited storage requirements and therefore can run on less expensive machines than database servers.

Since databases are quite large, they are partitioned across multiple servers. For good performance, most transactions need to execute entirely on one partition. Therefore, each transaction needs an application-supplied parameter that the mid-tier can map to the database partition that can execute the transaction. Windows Live services use three different partitioning schemes, based on hashing, value ranges, or table-look-up. Hash-based and range-based partitioning are commonly used in database systems, especially when the servers do not share disks. In a table-look-up-based scheme, each parameter value, such as a user-id, is explicitly mapped to a particular partition. The table-look-up scheme offers several advantages over the other two. First, it gives fine-grained control over load balancing. For example, when a new user-id is assigned, the mid-tier can assign it to a lightly-loaded server. Second, it can offer unique features for each user. For example, users who pay an annual fee receive special services not available to users who opt for free services. And third, it enables the system to minimize the time a user is affected by a schema upgrade by upgrading users one-by-one. By contrast, with hash-based and range-based partitioning, while a partition is being upgraded, the application would not know whether to access a user's data in the partition using the old or new format. Thus, all of the users in a partition would be inaccessible while the upgrade was in progress. Of course, these benefits come at a cost, namely, a database of <parameter-value, partition-id> pairs. For large services, the database may store hundreds of millions of parameter values. A case study of the table-look-up scheme appears in [5].

Each database partition is replicated for high availability. If a database server fails, its query load is absorbed by the replicas of the failed server's primary database partitions. If a database server needs to be taken off-line (e.g., because it is failing too frequently), its queue of committed updates must be drained to its replicas, so that its query and update load can be absorbed by its replicas.

3 Using Commodity Servers

The main sources of expense in running a large-scale service are hardware, data center infrastructure, power, networking, and system managers. Given the cost of premium servers as described in Section 1, one approach to reducing the expense is to move from such servers that cost (say) \$40K to commodity servers that cost (say) \$4K. This is surely beneficial, but it must be accompanied by attention to the other factors: the data center and people requirements. Some of these expenses are proportional to the number of machines and hence increase with a larger number of cheaper machines. The increase is usually sublinear and may be logarithmic for well-managed systems with excellent system management software. In this section, we discuss the factors that affect total cost and walk through a hypothetical calculation. Our goal is to give a feeling for the importance of certain factors and for the ratios involved. Our numbers and ratios depend on many factors, so they are merely rough estimates.

The data center is a major cost. Some of the cost factors are floor space (i.e., the building), communications, power, and cooling. Often these factors are in conflict. For example, a rural location might offer inexpensive real estate, but might have higher constructions costs and require new fiber for sufficient communications bandwidth. The server configuration can be designed to minimize its footprint and hence its real estate and construction cost. Power is an expense that can be controlled by moving the data center to a cheaper source of power, by using better cooling techniques, or by configuring servers that require less power. The latter usually lowers cooling requirements too.

Power is not only a cost but also a constraint. A large server installation can require tens of megawatts of power, which limits the locations where the installation can be built. It is therefore not surprising that Google, Microsoft, and Yahoo! have recently been building data centers near large hydroelectric facilities [4].

On the other hand, many installations have been set up in locations with excellent network connectivity. But these are often in locations where power is expensive. The move to rural locations with plentiful inexpensive power may be in part a response to the industry deciding to trade off power costs for networking costs.

System management is a major expense. System managers are responsible for monitoring load, diagnosing malfunctioning components, replacing failed components, and upgrading software to new releases. Job skills range from technicians who swap boxes to third-tier support engineers who diagnose subtle problems of performance, reliability, etc.

Given these considerations, let's do a hypothetical cost-benefit analysis of moving from premium to commodity machines. Consider a write-heavy application that is currently using premium four-way multiprocessor machines including RAID storage, and replicating each database partition for high availability. Using such premium machines, server costs dominate at about 50% of the total expense. About 25% of the cost is floor space, and less than 10% is people cost.

Getting a comparable level of reliability on commodity machines entails replicating each data partition on, let's say, four machines. Given that the commodity machines use slower disks, we might need as many as eight of them to replace each premium machine. This 8:1 ratio gives us more processor and memory capacity than we had before. But it might nevertheless be needed to get enough I/O capacity, especially if there's insufficient locality of reference to benefit from more caching.

Some networking costs are proportional to the number of ports required. Since we increase the number of machines, we potentially increase the number of network ports that are needed. Avoiding this increase requires some adjustment in the hardware configuration so that each rack has its own switch, which in turn connects to the main switch connected to the Internet.

The biggest effect of moving to commodity servers is people cost. Given the much larger number of servers and their lower reliability, there are many more system failures that require attention. Therefore, in our hypothetical cost analysis, system management explodes to 50% of total cost, which actually increases by 45% over the premium server configuration. The cost breakdown is 25% for servers, 15% for floor space, and 50% people.

Clearly, the solution is to automate many of the system management tasks, such as detecting faults, logging them, and analyzing logs to identify servers that should be reimaged or taken out of service and replaced. With the right tools, people costs can be brought down to the same level as with the premium server configuration. The tools have some cost to develop and maintain, but for large configurations this is small relative to the savings in systems management. As a result, total cost is now 25% lower than the premium configuration.

Another metric that addresses the same issue is the number of machines a system manager can handle. Suppose the fully loaded cost of a system manager is \$200K/yr. To get the 25% lower total cost, we need system management cost to be about 25% of the server cost. If servers are depreciated over 3 years, that means a system manager must be able to handle \$2.4M worth of machines, or about 600 machines. If a typical machine fails once a month, that means a system manager needs to handle 20 failures per day. Clearly, substantial automation is needed.

4 Performance Challenges and Solutions

Good performance is important both for customer satisfaction and to reduce hardware costs. In a large-scale application, the importance of many standard issues is magnified, such as the size of each attribute, the choice of indexes, disk layout, and the size of main memory. In this section we discuss some performance issues that are less widely known but also very important.

Replication: Each server holds primary copies of some partitions and replicas of other partitions. When a database server D fails, the database servers holding D's replicas need enough spare processor and I/O capacity to handle D's load in addition to their own. This extra capacity is part of the cost of high availability. To reduce this cost, each database partition that is being replicated is further partitioned into sub-partitions. Each sub-partition is replicated to a different machine. This reduces the headroom required on each machine to handle the additional load due to a failed primary partition. Thus, the more partitions the better. However, the sub-partitioning has a cost, namely, the fixed cost of I/O, memory and processing that the replication system consumes to support the additional replicas. For each replica, the primay has some partition-specific control information, an additional storage area to store commands to be forwarded to the replica, and an agent to do this work. These costs are independent of the number of updates. For an application with an update intensive workload, four sub-partitions has proven to be a good compromise between saving the cost of too much headroom and incurring the cost of sub-partitioning during normal operation. If the update workload is light, then less of the database server's spare capacity is needed for processing updates, so more of it can be used to handle a larger number of sessions. Hence, a larger number of partitions can be used in this case.

Replicas might also be useful to pick up some of the query load when the primary is overloaded and the machines that host its replicas are not. However, it is worthwhile to implement such a load-balancing mechanism only if such a skewed load is common. Since an important goal of partitioning tables is to spread the load evenly across partitions, most applications do not experience sustained skewed load. Hence, in most cases there is not much value in using replicas to handle queries except in failover situations.

Some applications have fan-out queries that access all partitions. As the workload increases, one can scale-out such an application by adding more servers and placing fewer partitions on each server. Although this increases the number of sub-queries required to execute a query, each sub-query has less work to do and there is more processing power to run them, so the overall system capacity is increased. If each server has just one partition, then one cannot scale out any further this way. So the next step is to split partitions into sub-partitions and place each sub-partition on a separate server, which enables more servers to be added. At some point, spreading the load in this way does more harm than good, since it may replace sequential I/O (to access a large partition) by random I/O (to access several sub-partitions albeit on different servers). In addition, there's a fixed overhead to run a query on each server and return its results, which increases with the number of servers. If the data is static or if reading data that is slightly out of date is acceptable, then other solutions may be preferable.

Session Management: In the multi-tier architecture described in Section 2, each mid-tier server needs a database session to each database partition that it accesses. These sessions consume resources. In our experience, a high-end commodity machine (as in Section 1) running Microsoft SQL Server^{TM} can handle up to several thousand database connections. As the number of connections increases much beyond that, the server can become unsteady or even unresponsive.

For example, suppose each database server has d databases and there are x mid-tier servers. If each mid-tier server has a session to every database, then each database server has $x \times d$ sessions. If a large scale web service has a thousand mid-tier servers with 4 databases per server, then each database server's session load is too high.

One way to avoid this problem is to partition the mid-tier servers so that each partition has sessions to only a small subset of the database servers. This greatly reduces the number of sessions to each database server. To do this, an application-supplied parameter is needed to direct each transaction request to a particular mid-tier partition, not just to a particular database server as described in Section 2. Front-ends should randomly select a particular mid-tier server in a partition, to spread the load evenly among them.

Suppose that a mid-tier server creates additional sessions to a database server that seems sluggish. This enables the mid-tier server to have more active database requests and hence continue to handle its load despite that database server's sluggishness. If the sluggishness is temporary and the extra sessions are dropped when they're no longer needed, then this might be a good strategy. However, if the sluggishness lasts long enough that all x mid-tier servers experience it, then they all create more sessions to that database server, let's say n sessions per database, which implies $x \times d \times n$ sessions per database server. Even after partitioning the mid-tier servers, the number of mid-tier servers per database server (i.e., x) may be in the hundreds. So again each database server may have more sessions than it can handle. The same problem can arise after a database server failure, where the load on that server is transferred to replicas. At this point we have exceeded the capacity for which the system was designed, so we must throttle back the incoming traffic.

One approach to avoid this is to have mid-tier servers control the database server load. Each has only a small fixed number of connections (e.g., 2) to the database servers it needs. The mid-tier server can then manage a queue of requests that pass through these connections. If the mid-tier server detects a major degradation of a database server's response time, it lets requests back up on its queue to prevent the database server from thrashing.

Materialized Views: In many Windows Live applications, data entry transactions and queries need different keys for indexing and clustering the data they access. For example, suppose the Users table is partitioned by user-id. Users have multiple credentials, such as email address and fingerprint encoding. They also have some other attributes, such as date of birth, country, gender, and preferences. If we often search for user by a credential, such as email address, then we need an index that maps email address to user-id. If we use the database system's indexing mechanism to do this, we would need to have it maintain an index with each partition. Since it regards each partition as a separate table, this doesn't help much; to access an email address, we would need to probe the index in every partition. Rather, we want the index to be partitioned by email address, so that given an email address we can access the unique index partition that stores it. We can do this by creating a table with two columns, email address and user-id, which is partitioned by email address. It still requires two database accesses to retrive user information, first to find the user-id for an email address and then to access the Users table. But this is the same disk access cost as would be incurred if the database system implemented this itself. The consistency of this table is maintained outside the database system. If the data isn't too critical, we can just choose a known order to write the information and live with the lack of transactional consistency guarantees. Otherwise, we can take the performance hit and wrap it in a distributed transaction. Most consumer services will opt for higher performance.

This works well when the index has low fanout (e.g., one record for each key value in the previous case). For larger fanouts, a materialized view may be needed. For example, an entertainment application typically captures movie reviews for each user. Since each user wants to be able to see all of his or her reviews, the table of reviews is partitioned by user-id. Users also want to read all the reviews of a given movie. Using the above approach, we can create a table that maps movie name to the user-id's of users who have written reviews. For each movie, there may be many such users, which are scattered across many partitions of the movie reviews table. Thus, many disk accesses will be required to retrieve them. To avoid this expense, we can have a second copy of the movie reviews table that is partitioned by movie. This is a materialized view where all of the reviews for a given movie are clustered together. Other examples are classified ads where ads are clustered by advertiser for data entry and by product category for querying, and shopping where products are clustered by vendor for data entry and by product category for querying.

Reducing checkpoint overhead: A database checkpoint flushes dirty buffers to disk, to reduce recovery time after a failure and to permit log truncation to avoid log overflow. Checkpoints can consume a lot of I/O bandwidth, so it is important to control their frequency and duration.

There are four times that Microsoft SQL Server initiates a checkpoint: (1) as a result of a DBA command such as backup and restore, removing a log file, taking the database online/offline, turning off database mirroring, etc.; (2) when the log becomes too long to allow recovery within the "recovery interval," which is the maximum

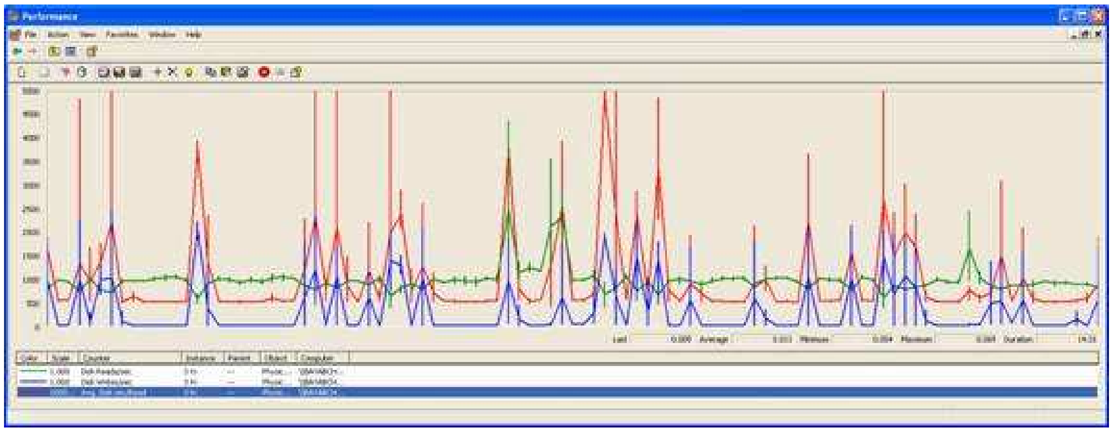


Figure 1: I/O operations on the database before reducing the number of checkpoints

recovery time specified by the database administrator (DBA); (3) when it is in “automatic log truncation mode” and the log is 70% full; and (4) when a DBA procedure calls the checkpoint command, which specifies a checkpoint duration, the amount of time for the checkpoint to complete. The latter enables SQL Server to calculate the number of writes per second required to flush all of the dirty pages in cache.

Many applications require frequent checkpoints, such as one per minute, to ensure fast recovery from failure and hence high availability. However, we found that in many applications taking checkpoints every half-hour is good enough, for three reasons. First, there is no risk of overflowing the log because log backups are taken every half-hour on the primary databases, after which the log is truncated. Second, as explained earlier, we ensure high availability by using replication, so fast log-based recovery is not crucial. Third, the log file generated in a half-hour interval is very small (approximately 20MB). If each log record is 200 bytes, then the log references 100K database pages. At 100 random I/Os per second per disk and, say, 20 disks, the recovery time is about a minute. So even if log-based recovery is needed, we expect it to be fast.

During a quick experiment on a production server, we set the recovery interval to a very large value. This eliminated automatic checkpoints on the primary databases except those triggered by log backup (which happens every half-hour). We still saw some automatic checkpoints on the replicas, which we believe are due to the small log files allocated to those databases and hence could be eliminated. In Figure 1 the red line (i.e., with highest peaks) and green line (i.e., with highest plateau and fewest peaks) show the writes/second and reads/second (respectively) without the checkpoint reduction. Figure 2 shows them with the checkpoint reduction. Notice the reduction in the number of spikes, which represents the I/O bottleneck as a result of checkpoints being generated.

Reducing checkpoints to once per half-hour risks having more dirty buffers to flush per checkpoint. However, in the above experiment, the number of buffers flushed per checkpoint did not increase when the inter-checkpoint interval increased from 1 to 30 minutes, for two reasons. First, in between checkpoints, a lazy writer is writing some of the dirty pages to make clean buffers available in response to memory pressure. An 8GB machine using 8KB/page has nearly 1M database pages in RAM. At 1000 writes/second, the lazy writer will write 1.8M pages in 30 minutes, which is almost twice the number of pages in RAM. So infrequently updated pages will certainly be flushed during that period to free up space for reads and will not require flushing at checkpoint time. The second reason is that during the half-hour, hot pages are written many times but only need to be flushed once at checkpoint time, so the large number of writes to these hot pages do not add to the checkpoint load.

A second checkpoint optimization is to spread each checkpoint’s writes over a longer time period, to avoid overwhelming the I/O subsystem. One approach is time-based. We assign a checkpoint duration that paces checkpoint I/Os to finish exactly within our half-hour time limit. For example, if a server has 8 databases, each database checkpoint should finish within 30/8 = 3.75 minutes. If each checkpoint writes about 15,000 buffers,

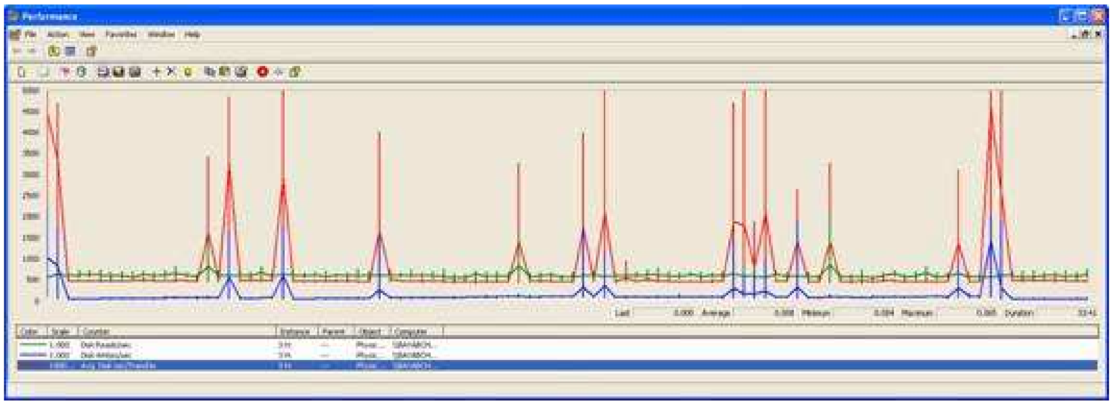


Figure 2: I/O operations on the database after reducing the number of checkpoints

then it will issue 67 writes/sec.

An alternative approach could be quota-based, where the user assigns a maximum I/O rate for each checkpoint and the database system ensures that this quota is not exceeded. When a checkpoint reaches the quota, it stops writing for a short time to avoid using too much I/O bandwidth. This guarantees that a checkpoint will not degrade the I/O rate available to applications below the given threshold, even when the checkpoint load is high. The cost is no upper limit on the checkpoint duration.

5 Research Approaches and Opportunities

The design of large-scale on-line systems has undergone continual change since their introduction in the late 1960's for airline reservation. The growing popularity of large-scale web services has changed things once again. These systems need to scale out to very large databases and user populations, often with hundreds of millions of users. They need to offer a mixture of transaction processing, document storage, and search functionality. They need to make the best use of the latest technology, inexpensive disk capacity, main memory, and flash memory. In particular, they need to cope with the changing hardware performance ratios, notably that disk capacity is increasing much faster than bandwidth and operations per second. And given the rapidly declining cost of hardware, they need to optimize for other major cost factors, notably system management. These factors present a challenge to designers of large-scale web services. They also present opportunities to researchers to discover ways to improve performance, availability, manageability and ease of programming, and to reduce cost.

References

[1] "A Conversation with Phil Smoot," $ACMQueue$ 3, 10 (Dec 2005), pp. 16-24.

[2] Shahine, O., "Inside Hotmail," http://channel9.msdn.com/ShowPost.aspx?PostID=39016, August 21, 2005.

[3] Transaction Processing Council, www.tpc.org.

[4] www.datacenterknowledge.com

[5] Xiong, M. B. Goldstein, and C. Auger, "Scaling Out SQL Server," $DellPowerSolutions$, Aug. 2005, http://www.dell.com/downloads/global/power/ps3q05-20050100-Auger.pdf.

Content, Metadata, and Behavioral Information: Directions for Yahoo! Research

The Yahoo! Research Team [*]

Abstract

In mid-2005, Yahoo! Inc. began an ambitious program to create a world-class industrial research lab focusing on how to deliver services over the web to a range of stakeholders, including advertisers, site owners, content publishers, and users. The resulting organization, Yahoo! Research, has embarked on a number of research directions. In this document, we report on these directions, with a particular focus on the data engineering problems we see as critical to our mission.

1 Introduction

Yahoo! Research (Y!R) sits at the nexus of more than half a billion unique users per month, several terabytes per day of data of various forms, several billion dollars of advertising revenue, and over one hundred distinct properties offering capabilities ranging from email, online news, and Web search to classifieds, personals, and horoscopes. Our goal is to shape the future of the internet, and we are focusing on a number of key research directions that we consider to be foundational.

Within Y!R, there are teams focusing on five research areas: *search and information retrieval, machine learning and data mining, microeconomics, community systems*, and *media experience and design*. Most projects draw from several of these five areas collaboratively. Common themes have emerged in these projects, many of which center around a combination of massive datasets and the Web as the delivery channel. In this overview, we highlight some key research directions rather than cataloging activities in each focus area, in order to reflect the synergy across areas at Y!R. In subsequent sections, we will cover work taking place in *platforms, advertising* and *search.*

Before describing the work itself, we will briefly describe the environment in which we see Yahoo! operating in the future. Figure 1 illustrates the situation. One may view Yahoo! as a match-maker that aggregates user attention by providing a wide range of high-quality content and community. We collaborate with content providers and site owners to accomplish this, and we then sell differentiated streams of user attention to advertisers, whose dollars flow to Yahoo!, as well as our partners, in order to fund the next round of improvements to the content and user-experience. We view all four stakeholders as central to our research initiatives, and we will give a more detailed perspective on how each research direction allows them to engage more effectively.

Within this environment, we see three overarching trends that we believe will shape the future of online interactions:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]Contact authors for this article, all at Yahoo! Research, are Raghu Ramakrishnan, Andrew Tomkins, and Ravi Kumar

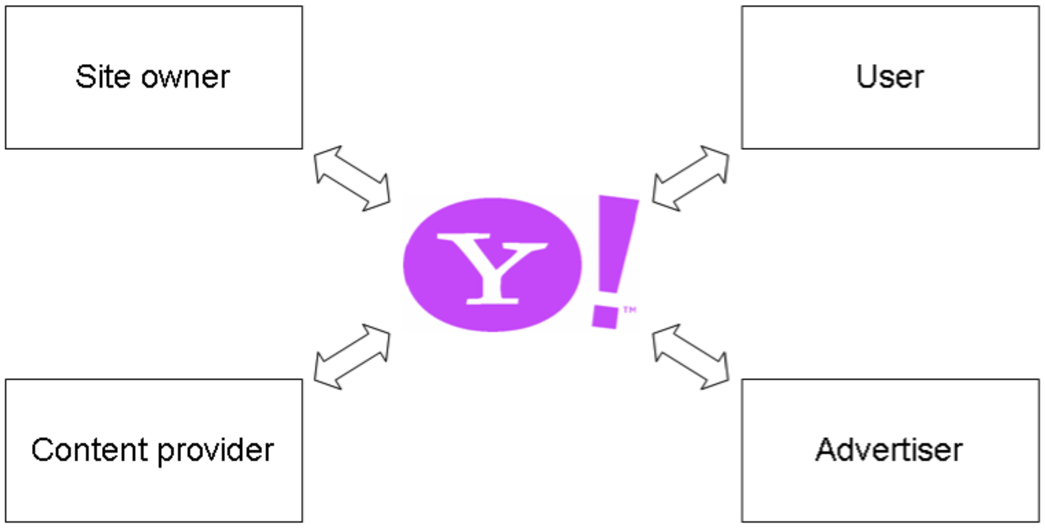


Figure 1: Key players in the Yahoo! ecosystem.

The emergence of structure. The highly heterogeneous data types we see range from text and html documents to query logs, user profiles, vertical content, listings, many forms of advertisements, user interactions, images, and video. In all these cases, there are distinct and interesting types of structure that can be useful both within and across datasets. We believe that capturing and exploiting such structure is a key to next-generation Web applications, including search and advertising. While exploitation of structure is a natural topic of discussion, we observe that capture of structure is also nontrivial: in many cases structure is either user-provided (and sometimes designed to mislead) or automatically extracted (and potentially error-prone).

The use of many different types of large-scale data could benefit from additional, integrated structure. First, existing content repositories often do not expose structure explicitly (e.g., individuals' home pages often contain contact information, but embedded in free text), and even when they do, rarely conform to uniform metadata standards (e.g. *price* on one shopping site might correspond to *cost* on another), and are not readily interoperable. Structure extraction technology ranging from entity extraction to site wrappers is currently applied only to a limited degree; it is likely this will change. Second, we have massive assets of user behavioral information that could significantly improve the user experience. But this information is not yet fully exploited due to data scale, the imperative to safeguard privacy, and the need to marry increasingly sophisticated user modeling to application logic. Finally, we see a growing stream of user-contributed content becoming a key resource. In many cases, it conforms to a large and growing set of schemas meant to incorporate specifics of either data type (question/answer pairs, new articles, blog posts, Web pages), or domain data (auto repair, medical information, etc). As a variant of user-contributed content, we see user-contributed metadata in the form of reviews, tags, bookmarks, ratings, or even clicks.

Design and dynamics of social systems. Online communities are a fundamental and increasingly important part of the web. A new science that brings to bear the mathematics of social networks, an economic theory of online interactions, and user experience design is required to further our understanding of how communities form and evolve, how we can facilitate their interactions, and how we can learn from shared community activities to enhance Web applications [8]. Increasingly, online applications in diverse domains must incorporate community tools simply as table stakes, and the richness of the baseline community offerings is growing rapidly. We routinely see bulletin boards with presence and real-time chat, tagging and folksonomy data arrangement, rating and reputation systems, and other such capabilities, on a wide range of sites. Further, as Web application development methodologies mature, it becomes easier for developers to "drop in" such capabilities. Today, it is simple to grab modules providing rich community semantics based on a local MySQL instance. But already the seeds are visible for hosted application providers that make such capabilities available, with potential advantages such as single authentication, integrated payment systems, and shared personalization. We view Yahoo! as a natural focal point for this evolution.

The Web as a delivery channel. The Web has become a powerful and ubiquitous means of delivering a range of end-user (e.g., email) and collaborative (e.g., IM, Y! Answers) applications to hundreds of millions of users. As online application development moves in the direction of "mash-ups" of online APIs (to a wide range of capabilities that may be combined into an application), the requirements on the backend change substantially. This has created a radically different approach to developing and distributing applications, disrupting the traditional software distribution model. In turn, it has challenged us to develop new types of service-oriented software platforms, new kinds of customizable application environments, and forced us to think about massively distributed systems with novel quality of service guarantees, fail-over mechanisms, and the ability to manage massive numbers of application instances. Furthermore, the landscape shifts very quickly, making it difficult to settle on business models and interaction paradigms. For example, advertisers are only now beginning to understand per-impression advertising (in which an advertiser is charged when a user loads a page containing a certain ad), and already, AJAX and related technologies are calling into question the entire notion of pageviews, as different parts of a page may update asynchronously in response to user activity or exogenous events.

With these themes in mind, we proceed as follows. Section 2 describes key directions in platform design and development. Then Sections 3 and 4 describe advertising and search respectively, which are key consumers of platform technologies; we conclude in Section 5.

2 Platforms

Data is central to Yahoo!s business. Some common workload patterns are:

- **Nugget storage/retrieval.** User-facing properties constantly store and retrieve data nuggets. In some cases all that is required is `get` and `put` based on a key, e.g. *store/retrieve Bob's stock portfolio*. Increasingly, more complex retrieval tasks are required, e.g. *get Bob's friends' favorite restaurants in cities on Bob's travel itinerary*.

- **Canned large-scale processing.** Many properties additionally rely on large-scale preprocessing of massive data sets. The obvious example is search, which performs link analysis over the many-billion page Web graph, to generate features used in ranking. However, many other Yahoo! properties also rely on large-scale data preprocessing, e.g. classification, clustering, entity extraction or wrapper induction over large corpora of webpages, news or advertisements. Typically, the same processing is run periodically, with every new version of the data set. Some cases require very short periods, in the limit becoming continuous queries over streams.

- **Ad-hoc large-scale processing.** Yahoo! analysts are constantly poring over our data sets, looking for patterns to inform the design of new products (and tune existing ones). Data sets of interest include Web crawls, click streams, advertisement entries, and various derived data. These "R&D queries" tend to be ad-hoc, and make heavy use of user-defined transformations, aggregation, and in some cases joins.

While these workloads are reminiscent of ones that have been studied extensively in the past, the sheer magnitude of the data creates new challenges. Extreme parallelism is the name of the game. We have two research initiatives on this front, one aimed at storage/retrieval workloads (Section 2.1) and one aimed at large-scale processing workloads (Section 2.2).

2.1 High-throughput storage/retrieval in an enormous database

We are investigating challenges underlying a very parallel storage/retrieval service that can support indexes for social-search and geo-search, and handle the types of workloads that arise in delivering a range of advertising, structured search, and community-oriented applications.

As we stated earlier, the Web has become a channel for delivering a range of end-user and collaborative applications to many users. Where users and organizations once had no option but to buy and install a stack of software ranging from database systems to middleware to specific applications, and to provision and maintain complex hardware and software installations, now they have the choice of going to a Web site such as Yahoo! or SalesForce.com and simply creating an instance of the application they want (e.g, an email account, a new Yahoo! group, or collaborating online in a social networking site such as Flickr or Y! Answers.)

The growing popularity of the application-as-service model makes it inevitable that more and more applications will become available as online services. This is a disruptive change to the way applications, and in particular data-driven applications and database management systems, have been traditionally designed. Now, we are challenged to develop systems that must be incredibly inexpensive on a per-application instance basis (the revenue from a Y! group cannot pay for dedicated hardware!), be easy to manage (a few operations people must oversee hundreds of thousands of groups), and very robust (since failures potentially affect thousands of application instances). One way to see this challenge is that we must build systems that allow us to be cost-effective database administrators (not to mention application developers and maintainers!) to the world, rather than vendors of DBMS software. On the other hand, since a given installation is intended to support many instances of a given application template, we have extensive knowledge of the workload, which is likely to be comprised of large numbers of requests (updates or queries) drawn from a relatively small set of request types. Similarly, there is often considerable latitude in the kinds of inconsistency that can be tolerated on specific requests.

Building systems to provide online applications as a service is one of the most intriguing challenges faced by the database community in many years. It requires us to leverage the lessons we have learned from designing parallel [3] and distributed database systems [9, 5], and to develop a new class of massively parallel, self-tuning, robust systems. It is likely that we will have to rethink almost ever aspect of database systems — What kinds of data? What mix of relational and text-based ranking queries? What models of concurrency and consistency? — to achieve the performance criteria required. A number of internal storage solutions at Yahoo! address these issues; see also [1].

2.2 Analytical processing over enormous data sets

For queries that perform wholesale analysis over data such as web crawls and search query logs, we focus on intra-query parallelism. The higher the degree of parallelism, the faster the response time for individual queries, which is critical for continuous queries and for ad-hoc R&D queries.

In principle, if we have a data set of size $|D|$ to analyze, and we divide the data and processing among n nodes, then each node need only handle around $|D|/n$ data. Grouping or joining may require repartitioning the data, which only doubles the amount of data each node needs to handle. In general, if data is to be repartitioned k times, we expect $(k+1) \cdot |D|/n$ units of work per node. Hence for a given query, doubling the number of nodes should halve the response time.

Unfortunately it is not possible to achieve this ideal scale-up behavior in practice. Moreover, we are finding that parallel query processing techniques that exhibit near-ideal scale-up when $n = 10$ or perhaps $n = 100$, do not continue to do so when $n = 1000$ or more. We are studying new architectures and algorithms aimed at good scale-up for $n = 1000$ or even $n = 10,000$, so that we can answer ad-hoc queries over multi-terabyte data sets in minutes [2, 7, 4, 6]. We expect this capability to be a key enabler of R&D activity and business intelligence applications going forward.

As our platform becomes used to provide analytic services to a large number of internal R&D "customers," a second major challenge will emerge: intelligent physical design to optimize the overall workload. In the parallel data management space, physical design includes data partitioning, as well as the usual degrees of freedom with respect to the choice of indexes and materialized views. We expect this problem to be very difficult to solve, based on the experience in other distributed computing environments such as grid computing, but it is also very important. If we do not do a good job, then our "customers" will not accept physical data independence, and will spend valuable cycles tinkering with the physical design by hand.

3 Advertising

Web advertising spans Web technology, sociology, law, and economics. It has already surpassed some traditional mass media like broadcast radio and it is the economic engine that drives Web development. It has become a fundamental part of the Web eco-system and touches the way content is created, shared, and disseminated—all the way from static html pages to more dynamic content such as blogs and podcasts, to social media such as discussion boards and tags on shared photographs. This revolution promises to fundamentally change both the media and the advertising businesses over the next few years, altering a $300 billion economic landscape.

As in classic advertising, in terms of goals, Web advertising can be split into *brand advertising*, whose goal is to create a distinct and favorable image for the advertiser's product, and *direct-marketing advertising*, which involves a "direct response": buy, subscribe, vote, donate, etc., now or soon.

In terms of delivery, there are two major types:

1. *Search advertising* refers to the ads displayed alongside the "organic results on the pages of search engines. This type of advertising is mostly direct marketing and supports a variety of retailers from large to small, including micro-retailers that cover specialized niche markets.

2. *Content advertising* refers to ads displayed alongside some publisher produced content, akin to traditional ads displayed in newspapers. It includes both brand advertising and direct marketing. Today, almost all non-transactional Web sites rely on revenue from content advertising. This type of advertising supports sites that range from individual bloggers and small community pages, to the web sites of major newspapers. There would have been a lot less to read on the Web without this model!

Web advertising is a big business, estimated in 2005 at $12.5B spent in the US alone (Internet Advertising Board—www.iab.com). But this is still less than 10% of the total US advertising market. Worldwide, internet advertising is estimated at $18B out of a $300 total. Thus, even at the estimated 13% annual growth, there is still plenty of room to grow, hence an enormous commercial interest.

At Yahoo! Research we are exploring designs for next generation advertising platforms for contextual and search ads. From an ad-platform standpoint, both search and content advertising can be viewed as a matching problem: a stream of queries or pages is matched in real time to a supply of ads. A common way of measuring the performance of an ad-platform is based on the clicks on the placed ads. To increase the number of clicks, the ads placed must be relevant to the user's query or the page and their general interests.

There are several data engineering challenges in the design and implementation of such systems.

The first challenge is the volume of data and transactions. Modern search engines deal with tens of billions of pages from hundreds of millions of publishers, and billions of ads from tens of millions of advertisers. Second, the number of transactions is huge: billions of searches and billions of page views per day. Third, there is only a very short processing time available: when a user requests a page or types her query, the expectation is that the page, including the ads, will be shown in real time, allowing for at most a few tens of milliseconds to select the best ads.

To achieve such performance, ad-platforms usually have two components: a *batch processing component* that does the data collection, processing, and analysis, and a *serving component* that serves the ads in real time. Although both of these are related to the problems solved by today's data management systems, in both cases existing systems have been found inadequate for solving the problem and today's ad-platforms require breaking new ground.

The batch processing component of an ad-system processes collections of multiple TB of data. Usually the data is not shared and a typical processing cycle lasts from a few minutes to a few hours over a large cluster of hundreds, even thousands of commodity machines. Here we are concerned only about recovering from failures during the distributed computation, and most of the time data is produced once and read one or more times. Commercial database systems deployed on the same scale would be prohibitively expensive. The reason for this is that database systems are designed for sharing data among multiple users in the presence of updates and have overly complex distribution protocols to scale and perform efficiently at this scale. The backbone of Web data batch processing components (see Section 2.2) is therefore being built using simpler distributed computation models and distributed file systems running over commodity hardware. Several challenges lie ahead to make these systems more usable and easier to maintain. The first challenge is to define a processing framework and interfaces such that large scale data analysis tasks (e.g., graph traversal and aggregation) and machine learning tasks (e.g., classification and clustering) are easy to express. So far there are two reported attempts to define such query languages for data processing in this environment [6, 7]. However, there has been no reported progress on mapping these languages to a calculus and algebra model that will lend itself to optimization. Conversely, to make the task easier, new machine learning and data analysis algorithms for large scale data processing are needed. In the data storage layer, the challenge is to co-locate data on the same nodes where the processing is performed.

The serving component (see Section 2.1) of an advertising platform must have high throughput and low latency. To achieve this, in most cases the serving component operates over a read-only copy of the data replaced occasionally by the batch component. The ads are usually pre-processed and matched to an incoming query or page. The serving component has to implement business logic that, based on a variety of features of the query/page and the ads, estimates the top few ads that have the maximum expected revenue within the constraints of the marketplace design and business rules associated to that particular advertising opportunity. The first challenge here is developing features and corresponding extraction algorithms appropriate for real-time processing. As the response time is limited, today's architectures rely on serving from a large cluster of machines that hold most of the searched data in-memory. One of the salient points in the design of the ad server is an objective function that captures the necessary trade-off between efficient processing and quality results, both in terms of relevance and revenue.

In summary, today's search and content advertising platforms are massive data processing systems that apply complicated data analysis and machine learning to select the best advertisements for a given query or page. The sheer scale of the data and the real-time requirements make this a very challenging task. Today's implementations have grown quickly, and often in an ad-hoc manner, to deal with a $15B fast growing market. There is a need for improvement in almost every aspect of these systems as they adapt to even larger amounts of data, traffic, and new business models in Web advertising.

Looking beyond, in the context of auctions for internet advertisement, microeconomics and data analysis go hand in hand. Using the tools of microeconomics one can make theoretical predictions regarding how advertisers would respond to changes in an auction mechanism (such as changing reserve price or allowing advertisers to submit separate bids for ads targeted to different demographics). With careful data analysis one can estimate how such changes would impact Yahoo! and its users and advertisers. Doing this kind of data analysis is possible only with a high-performance advertisement platform.

4 Search

The Search focus area at Yahoo! is broadly concerned with research that enables users to satisfy an information need. The scoping is intentionally broader than Web search alone. Yahoo!'s content portfolio ranges well beyond simple Web pages, and the types of objects that must be retrieved are broadly heterogeneous, and in some cases extremely complex. For example, a search across a selection of publicly-accessible bulletin boards could return an appropriate message, a thread consisting of hundreds of posters and thousands of posts, or a board with thousands of members. In the near future, it may well return a much richer structure containing multiple overlapping embedded bulletin boards, multimedia content, questions and answers, news articles, perhaps an online marketplace, and so forth. Effective ranking over such complex objects is well beyond the state of the art.

Search technology is in flux. It is common for employees even at technically sophisticated organizations to have access to enterprise search tools that are markedly inferior to what is available on the Web, despite the much higher average quality of enterprise content. We are pursuing four key research directions, as discussed below.

4.1 User-generated metadata analysis

The first direction is classically responsible for the success of Web search. It involves the incorporation of social information: there are key cues (such as incoming anchortext) that allow internet search engines to employ the aggregate perspective of an enormous number of users in deciding which results to return. And we are currently at the center of a rapid shift in the nature of this social input. Historically, search engines have drawn primary leverage from hyperlinks and anchortext. In other words, most of the input has come from a relatively small number of elite Web users who are capable of authoring and serving html content. Today, however, the number of distinct users generating useful metadata is growing rapidly due to three factors. First, the emergence of simple Web authoring tools such as hosted blogging software makes it possible for authorship to migrate from the elite to a much larger base of online users motivated to express themselves. Second, the introduction of new models for explicit creation of metadata versus content, such as tagging and bookmarking (e.g., through `del.icio.us`), the creation of rich profiles (e.g., `myspace.com`), or even the creation and publishing of multimedia content (e.g., `youtube.com`) lowers the barrier from authorship to lighter-weight interactions like commenting on somebody else's content. And finally, there are situations in which content consumption itself is a generator of useful metadata. For example, when a search engine shows a list of results, the result clicked by a user is potentially helpful information. In the extreme, companies such as Google allow users optionally to share their entire browsing behavior with the search engine. The privacy implications of such sharing have not yet been fully explored, but if the social contract between users and online tools expands to include such activities then another order of magnitude of data scale becomes available to help us understand the quality of content.

4.2 Aggregate analysis

The second key direction in search is the aggregation of data and metadata across multiple dimensions, at multiple levels of granularity. By aggregation, we mean something more than simple rollups using straightforward algebraic functions; instead, we mean sophisticated analysis making use of all information available for a natural grouping of content objects. For example, we might consider all pages on a website, or all content related to medical matters, in order to extract common topics and classify content according to which of the common topics are being discussed. We will speak to the key forms of aggregation in one more level of detail.

First, there is aggregation at the level of *content sources*, for instance a website, community, or blog, which may be viewed as a cohesive generator of potentially useful content over time. Understanding the reputation, topical focus, authorship, affiliation, quality, and reliability of each content source becomes a key advantage in aggregating content from these sources. The aggregation may happen at multiple levels, such as a website and a key subdirectory or subsite of the website. The type of information to be aggregated is not limited to information extracted from the content itself. For instance, it is clearly useful to know if there are coherent groups of users who are the typical exploiters of a content source.

Second, in addition to aggregation at the level of content sources, there are also cross-source forms of aggregation, such as topical aggregation (as in our earlier example of analyzing all content around medical matters), target aggregation (for instance, analyzing the content which draws 14-16 year old male viewers), entity aggregation (all content regarding George Bush, or the Samsung HLS6187 high-definition TV), and so forth.

4.3 Cross-content analysis

The third trend is analysis to extract insights that are accessible only in the combination of multiple giant databases of disparate schemas. This form of analysis arises in two settings. We begin with the first and simpler setting, which is related to federated search or distributed information retrieval.

Queries to a Web search engine are primarily answered by providing a link to a relevant Web page, but increasingly, other sources are being federated in. For example, queries for local restaurants, or musical artists, or movies, or products already receive special handling, as do queries that may be syntactically identified as belonging to a special class, such as airline flights or package tracking identifiers. These tricks may be performed with minimal query processing overhead. Additionally, however, parallel queries to other corpora such as collections of images or video are being performed in Web search to determine whether other classes of content might be appropriate. Yahoo! already federates responses from its Answers corpus, and Google's Coop allows sophisticated users to generate plugins directing certain queries in parallel towards alternate user-specified corpora. Of particular interest to the data management community is the trend towards including structured records in results, as for instance in Google Base or Yahoo! shortcuts to shopping data.

Generally, the move towards retrieving results from numerous disparate corpora is well on its way. However, as yet there is no truly successful work on providing the user with a single cohesive set of results. The best of breed approach is simply formatting the result page so that it is clear which results come from which corpus, and employing only the simplest techniques to merge modules from each relevant corpus onto a result page.

Finally, there is a second setting in which cross-content analysis arises; this latter setting moves beyond the scope of finding results from multiple corpora, namely, the problem of simultaneous batch mining of multiple datasets in order to extract insights or results that are unavailable from any collection singly. For instance, the combination of Web content, site-aggregated metadata, click data, and publicly available user profile data might give a perspective on potential query responses that is far beyond the current model.

4.4 Community analysis

Increasingly, there is now widespread acceptance that meeting a user's information need on the Web is more than a matter of human-computer interaction: it is also a matter of human-human interaction. As an example, consider the remarkable success of "knowledge search," as provided by Naver in Korea and many other related offerings, both in Southeast Asia and increasingly elsewhere in the world (for instance, Yahoo! Answers is a global offering of this form). A user enters a question, which is routed to other online users who might be able to provide an answer. Typically multiple answers are received in well under a minute. Clearly in this case traditional approaches to indexing have little to contribute, and whole new questions around answerer competencies become relevant. Further, there is a completely new question regarding the fusion of online knowledge (either in traditional documents, or in other forms such as answers to prior questions) with human knowledge that resides in wetware but might be accessible at significant latency through the paradigm of asking.

Community analysis throws open a wide array of questions in microeconomics, including incentives and markets that emerge as a result of online human-human interactions. Microeconomic models can be used to derive effective reputation feedback mechanisms that are resistant to gaming. Experimenting with intelligently designed reputation feedback systems and carefully analyzing the data generated by such experiments can help to improve such mechanism and to identify ideas that work in practice as well as predicted by the theory.

5 Conclusions

In this paper, we have outlined three key trends in the future of online interactive media: heterogeneous structured content; online social systems; and web-delivered applications. Based on these trends, we have described a number of heavily data-centric research directions that Yahoo! Research will pursue, in the areas of platforms, advertising, and search.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. In *Proc. 7th Symposium on Operating System Design and Implementation*, pages 205–218, 2006.

[2] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. 6th Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

[3] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsaio, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. Technical Report MSR-TR-2006-140, Microsoft Research, October 2006.

[5] J. B. R. Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. Introduction to a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 5(1):1–17, 1980.

[6] C. Olston, B. Reed, R. Kumar, D. Meredith, U. Srivastava, and A. Tomkins. Querying enormous data with Pig, 2006. Manuscript.

[7] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel data analysis with Sawzall. *Scientific Programming Journal*, 13(4):277–298, 2005.

[8] D. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton & Company, 2003.

[9] R. Williams, D. Daniels, L. Haas, G. Lopis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, and R. Yost. R*: An overview of the architecture. In P. Scheurman, editor, *Improving Database Usability and Responsiveness*, pages 1–27. Academic Press, New York, 1982.

Structured Data Meets the Web: A Few Observations

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

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 articulate challenges based on our experience with addressing them at Google, and offer some principles for addressing them in a general fashion.

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 [4], the ESP game [14], Google Co-op [6]) that enable people to add labels to content (pages and images) on the web, and Google Base [7], a service that allows users to load structured data from any domain they desire into a central repository.

Google is conducting multiple efforts whose common goal is to leverage structured data on the web for better search and to help people create structure that aids search. This paper describes some of our initial observations regarding the problem and some of the challenges entailed by them.

To begin, we look at what kinds of structure can be associated with data on the web. As is typical when managing structured data, there is a trade-off between the investment in creating the structure and the benefit obtained from having the structure. On the Web, the trade-off is a trickier one, because we are trying to appeal to a much larger audience, both to create the structure and to benefit from it. We discuss how the different efforts fall w.r.t. this trade-off.

Next, we consider how structured data is integrated into today's web-search paradigm that is dominated by keyword search. We argue that a critical aspect of any use of structured data on the Web is that it be seamlessly integrated with standard web search. We discuss how our different efforts address this challenge.

Finally, we pinpoint (what we believe to be) the key reason that makes querying structured data on the web so challenging. Specifically, on the web we model *all* domains of possible interest to users. We cannot make any assumptions about the scope of the domains for which we have structured data, and because of that, creating a *schema* is out of the question. We describe some of the challenges involved in providing access to *data about anything*, and some of our initial steps addressing these challenges.

2 Structured Data On the Web

In this section we describe three kinds of structured data that exist on the Web today. We then discuss the level of structure for each kind of data and how it is currently integrated into Web search.

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 content on the deep web is believed to be possibly more than the current WWW, and typically is of very high-quality [1].

A simple count of the number of forms on the Web would be very misleading if our goal is to understand how much deep web content exists. For instance, there are numerous forms that appear on a large number of web-pages, but direct the user to the same underlying data source. A significant number of forms on the web are simply redirects of site-specific searches to a major search engine. Another large source of forms involves Web 2.0 applications, where users contribute data to online communities. Further, forms are also used for applications such as logins, subscriptions, feedback, etc., none of which yield useful structured data. In [12] we offer some evidence from our explorations at Google that predict the number of deep web sources (that do not fall into the above categories) to be in the tens of millions.

The content on the deep web varies wildly. 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 [13]. Further, sources vary from those offering only keyword queries through a single text box, to those with detailed forms with many select-menus and other refinement options.

2.2 Google Base

The second source of structured data on the web that we discuss, Google Base, is an attempt to enable content owners to upload structured data into Google so it can be searched. 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 about clinical trials, event announcements, exotic car parts, people profiles, etc. Google indexes this data and supports simple but structured queries over it.

The data model for Google Base data is purposely kept very simple. 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, model = Civic, location = San Francisco, CA, etc. 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.

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 well over 10,000 item types that together contribute almost 100,000 unique schemata (unique set of attributes names associated with some item). There are over 1000 item types that have more than 10 items. In addition to the sheer number of item types, we observe two other important aspects of Google Base data. 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, that is typically not considered in discussions of data heterogeneity – users provide different item-type names or attribute names deliberately in an attempt to improve the ranking of their items. We are thus witnessing a kind of database design by the masses.

To get a feel for the diversity of data at the attribute-level, we looked at the items of type vehicle, and found that there are over 50 attribute names that occur in 10 or more items. While this might seem to be a large number, there is a smaller core set of attributes that occur in a large number of items. This includes common attributes like make, model, location, color, and price. A commonly occurring 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 there are separate attributes.

Finally, we note that Google Base data also displays significant heterogeneity at the level of attribute values. For example, considering the different values for the attribute color of vehicles, we 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 (colors of cars in this case) makes it possible to perform such reconciliation which would not be possible for purely unstructured data.

2.3 Annotation Schemes

There is a third class of structured data on the web which is the result of a variety of *annotation schemes*. Annotation schemes enable users to add tags describing underlying content (e.g., photos) 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 [14] 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. The important point to note is that the annotations are very light-weight – we only annotate which facet the page belongs to and nothing else. 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.[1] Google Co-op uses a collaboration model to manage annotations contributed by different users – the creator or a customized search engine approves a list of *collaborators*. Only the annotations contributed by collaborators get included in the customized search engine.

2.4 A Spectrum of Structure Levels

Data on the web, and in particular the kinds of data we described above, vary significantly in its level of structure. The trade-offs involved in structuring data are quite obvious: it takes an effort to create the structure (i.e., create schemata, fit the data into the schema, run a database system), but the benefit is more powerful querying capabilities.

On the Web, this trade-off becomes even more critical. First, our goal is to enable a much broader set of people to create structured data (e.g., in Google Base and with annotation schemes), and therefore the process must be very easy. Second, our user base is incredibly diverse, and therefore querying needs to be easy, structured data needs to be seamlessly integrated with unstructured data, and when structured data appears, it must be easy to explain to the user how to interpret it or query it further. Third, the Web offers us some novel points on the structure spectrum that are valuable to explore. We now examine each of the three cases mentioned above.

The deep web: The deep web represents the most structured data on the web. Although many forms offer search over document collections, the majority of forms offer search into data that is stored in back-end databases and the form queries get translated into queries on the database. Hence, the deep web represents a point in the spectrum where the content creators invested most effort.

Ironically, in many cases, the return on investment for deep web content creators is low. In the early days of the web, high quality collections of structured data were very few, and therefore they became well known and thus were able to attract traffic. Today, however, there are too many high-quality sources, but since web-crawlers do not crawl past forms, the content is invisible to the major web-search engines (and hence to many users looking for the data). As a result, these sites get a fraction of the traffic that is relevant to them.[2]

Google Base: Google Base had to take a much more flexible approach to structuring data. As mentioned before, the idea is that any content provider can contribute structured data to Google Base, without having any expertise in data management. There are a few aspects in which Google Base makes it easier to create structure. First, the data model is kept very simple (objects fall into item types and have attributes). Second, content providers are not forced into any schema. They can choose item types and attribute names from what Google offers, but can invent their own. The same applies to values of attributes in the data. Third, the data can be as semi-structured as the content provider wishes. Specifically, a feed to Google Base can provide as few or as many attributes. Much of the information associated with a Google Base entry can be in unstructured form or even in a URL pointed to by the Google Base entry. As we discuss later, this relaxed approach to structure raises challenges for ranking algorithms.

Annotation schemes: Annotation schemes represent the other extreme of structured data on the web. They require the users to provide very minimal structure (specifically, only the annotations). In a sense, it is as if we attach to every piece of content an attribute isAbout and the annotations provide the values for that attribute. Clearly, annotations can be incorrect and heterogeneous, i.e., use multiple words to say the same thing. However, the minimal structure provided is already enough to offer improvements in search experience.

[1]There is a fallback to general web search if there are no results from the customized search engine.

[2]In some cases, sites have learned to materialize some of their content to enable easier access to web-crawlers.

3 Integrating Structured and Unstructured Data

The reality of web search characteristics dictates the following principle to any project that tries to leverage structured data in web search: querying structured data and presenting answers based on structured data must be *seamlessly* integrated into traditional web search. This principle translates to the following constraints:

- Queries will be posed (at least initially) as keywords. Users will not pose complex queries of any form. At best, users will pick refinements (by filling out a form) that might be presented along with the answers to a keyword query.
- Queries will be posed from one main search destination, e.g., the main page of a search engine. Users will find it hard to memorize the existence of specialized search engines, especially ones that they use occasionally. The onus is hence on the general search engine to detect the correct user intention and automatically activate specialized searches.
- Answers from structured data sources need to (as far as possible) appear along-side regular web-search results. Ideally, they should not be distinguished from the other results. While the research community might care about the distinction between structured and un-structured data,[3] the vast majority of search users do no appreciate the distinction and only care if the results meet their requirement.

Hence, a significant challenge for each of the efforts mentioned above is to fully integrate into web search. We now describe the approach that each of the projects take and the challenges they face in pursuing them.

Deep Web: 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 in that domain 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 as queries over all (or a subset of) the forms in that domain. In fact, in specific domains (e.g., travel, jobs) this approach is used to create vertical search engines. For general web search, however, the approach has several limitations that render it inapplicable in our context.

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 (as we describe in the next section). Hence, it is infeasible to design virtual schemata to provide broad web search on such content.

The second limitation is the amount of information carried in the source descriptions. Although creating the mappings from web-form fields to the virtual schema attributes can be done at scale (e.g., using techniques described in [3, 10, 11]), source descriptions need to be much more detailed in order to be of use here. Especially, 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, the high volume of traffic can potentially crash the sites. 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.

The third limitation is our reliance on structured queries. Since queries on the web are typically sets of keywords, the first step in the reformulation will be to identify the relevant domain(s) of a query and then mapping the keywords in the query to the fields of the virtual schema for that domain. This is a hard problem that we refer to as *query routing*.

Finally, 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 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

[3]In fact, even as database people investigate the issue deeper, it appears that the distinction is fuzzy to them as well.

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. Similarly, the query-routing issue 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 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.

Of course, surfacing has its 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. Still, at least the pages are in the index, and thus can be retrieved for many searches. Other disadvantages are that 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 significantly 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.

Google Base: Google Base faces a different integration challenge. Experience has shown that we cannot expect users to come directly to base.google.com to pose queries targeted solely at Google Base. The vast majority of people are unaware of Google Base and do not understand the distinction between it and the Web index. Therefore, it is crucial to access Google Base in response to keyword queries posed on Google.com. This leads to two main challenges:

- Query routing: given a keyword query, decide if there is data relevant in Google Base and map it to a set of relevant item types in Google Base.
- Result ranking: to fully integrate Google Base results we need to rank results across properties. Specifically, for the query "Honda Civic Mountain View, CA", there might be 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.

In some cases, the answer offered by Google Base may be a specific entry (or set of entries) in Google Base. But in most cases, the answer consists of a few example entries from Google Base and a form that enables the user to refine the search results. Hence, an additional challenge Google Base needs to address is that of proposing appropriate refinements. Google Base proposes a set of attributes with candidate values in select-menus. 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 select-menus for the attributes. With each successive user refinement, Google Base recomputes the set of further 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 select-menus values for color change to reflect only the colors of Honda Civics available in the Mountain View area.

Annotation Schemes: 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.

4 A Database of Everything

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.

However, the challenges described in the earlier sections lead us to the crux of the problem of querying structured data on the Web: 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. However, the issue is much deeper. Data on the Web encompasses much of human knowledge, and therefore it is not even possible 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 [2], 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, you need to maintain the representation in over a hundred languages.

Currently, several solutions are being deployed for addressing the needs of each of the existing services. In parallel with addressing these needs, we are developing some general concepts that will enable us to deal with such challenges in a unified way.

The key idea we are pursuing is that of a system that integrates data from multiple sources but is designed to handle *uncertainty at its core*. Specifically, uncertainty in such a system can have several sources:

- mapping keyword queries into structured queries on a structured data feed,
- mappings between the schemas of disparate data sources, and
- uncertainty about the actual data in the system, that may have been extracted from unstructured document using automatic techniques.

In the contexts described above, such a system will enable us to let many structures (i.e., schemas) exist but without the need to completely integrate them as required by today's data integration systems. Instead of necessarily creating mappings between data sources and a virtual schema, we will rely much more heavily on schema clustering. Clustering lets us measure how close two schemas are to each other, without actually having to map each of them to a virtual schema in a particular domain. As such, schemas may belong to many clusters, thereby gracefully handling complex relationships between domains. Keyword queries will be mapped to clusters of schemas, and at that point we will try to apply approximate schema mappings in order to leverage data from multiple sources to answer queries. The system we envision is an instance of a dataspace support platform [5, 9], where we structure the data in the system in a pay-as-you-go fashion. We offer some more detail about such a system in [12].

5 Conclusions

There are staggering amounts of structured data on the Web today, and we must finally address the challenge of leveraging such data and combining it with unstructured data. This paper described three main sources of structured data, the deep web, Google Base and a variety of annotation schemes, and outlined the main challenges involved in querying this data.

From the perspective of data management, we are used to thinking in terms of a dichotomy between structured data and unstructured data, and the Structure Chasm that lies between them [8]. Web-scale heterogeneity, i.e., data in millions of disparate schemata, presents a unique point between the two ends of the chasm. Our goal is to build a data management platform that 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. We believe that the most significant of these challenges is building a data management system that can model *all data*, rather than data that in a particular domain, described by a single schema.

References

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

[2] Cycorp, Inc. http://www.cyc.com/cyc/technology/whatiscyc.

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

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

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

[6] Google Co-op. http://www.google.com/coop.

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

[8] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the structure chasm. In *Proc. of CIDR)*, 2003.

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

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

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

[12] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Navigating the seas of structured web data. In *Proc. CIDR*, 2007.

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

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

A funny thing happened on the way to a billion…

Alfredo Alba, Varun Bhagwan, Mike Ching, Alex Cozzi, Raj Desai,
Daniel Gruhl, Kevin Haas, Linda Kato, Jeff Kusnitz, Bryan Langston,
Ferdy Nagy, Linda Nguyen, Jan Pieper, Savitha Srinivasan,
Anthony Stuart and Renjie Tang

IBM Almaden Research Center, San Jose, CA, USA

Abstract

IBM's Semantic Super Computing platform has been designed to ingest, augment, store, index and support queries on billions of documents. It faces a number of requirements and challenges not found in similar, smaller content management systems. We describe some of these challenges and lessons learned in the areas of solution design, hardware, operations, middleware, algorithms and testing.

1 Introduction

As document repositories approach the multiple billion document level, users are finding that simple search no longer meets all of their information retrieval requirements and traditional approaches to other content management functions are proving to be inadequate. As a result, research is focusing on new architectures and technologies that enable the development of the next generation of advanced information management applications.

Over the last 8 years we have built a Web-scale content management system [12, 8] to collect, analyze, store and serve billions of documents. In doing so, we have encountered a number of ways in which such systems must differ from traditional content management systems. Even though our initial focus was Web data, we've found that many companies have similarly large collections of data and information needs.

In this paper, we present several of our hard-won lessons for handling information storage, retrieval and analysis requirements at the multiple billion document level. These include new insights in the areas of information discovery, data sampling, performance management, system administration and operations.

2 System Considerations

2.1 Discovery is more than Search

Corporate document archives have rapidly moved to the multiple billion document level under the pressures of various regulatory and compliance rulings. As our clients begin to leverage increasingly valuable information out of these unstructured document archives, they are finding that simple search, while a valuable tool, is not

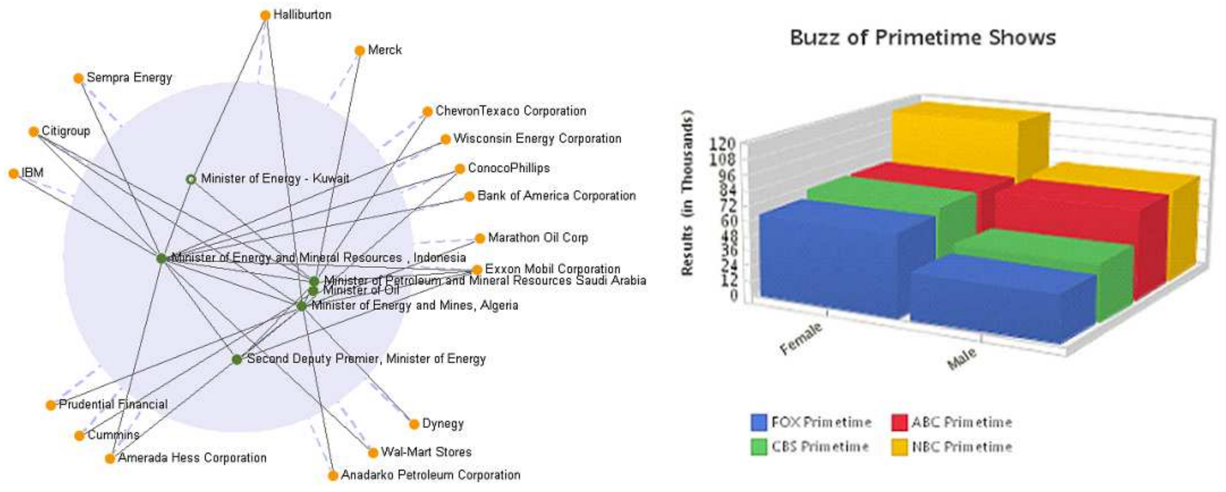


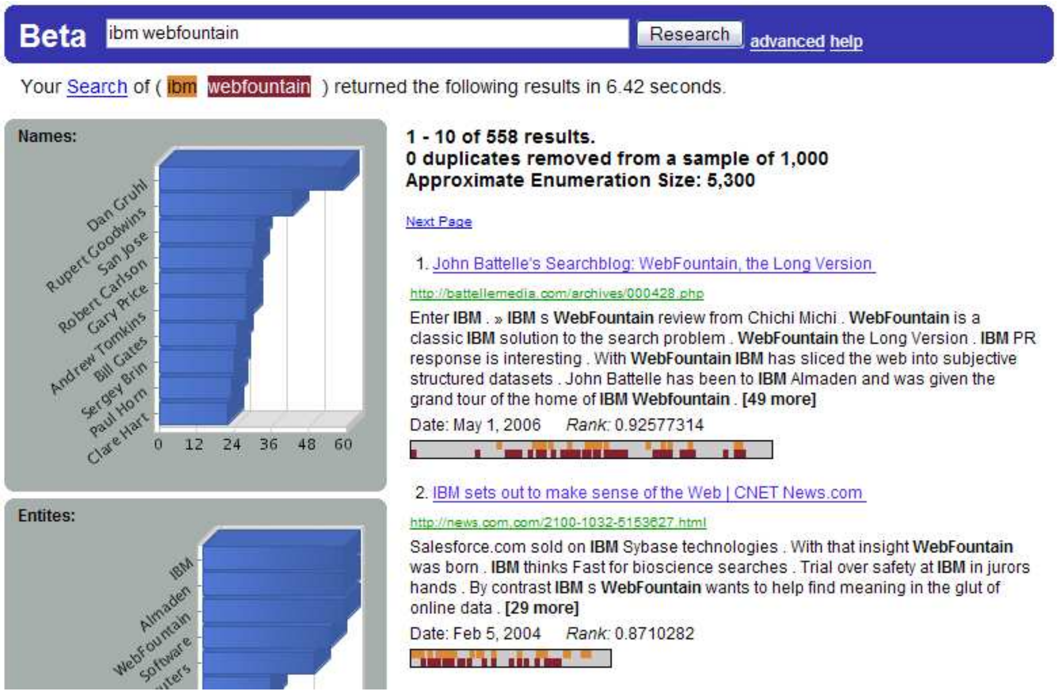
Figure 1: Discovery is more than search. An analyst may be interested in the names of companies most frequently mentioned with OPEC ministers, and the relationship between the ministers and the companies based on Internet data (left). Another analyst might be interested in the TV networks watched most by users of a popular on-line community site, broken down by male and female users (right). In both cases the information being sought is aggregated at a high level and presented in a business intelligence like manner - very unlike traditional search responses.

enough. Any one search request may match a very large number of documents. Simply listing the 150,000 hits is not an option. Analysts need tools similar to the "big picture" business intelligence reports common in the structured data domain.

We refer to such suites of tools as *Discovery*. Discovery applications support the production of charts, graphs, plots and other visualizations, as well as allowing drill-down to the source documents that support them (see Figure 1 for examples). They allow the definition of complex annotations and taxonomies, with subsequent application of such knowledge structures to arbitrary corpora. For example, if the corpus was a corporate email spool being examined for a compliance application, an annotator might be looking for something that could be construed as a stock tip. A pharmaceutical company might want to scour web data for potential patent violations. Discovery systems must support alerts or triggers, which run when new data enters the system or conditions change. Analysts need to be able to track a topic over time [14].

These more mature information needs impact the underlying infrastructure with a number of novel functional and non-functional requirements. There are certain query operators that are not common to the pure search domain, such as aggregation and reporting of accurate matching result cardinality (as opposed to the "estimates" that traditional search engines might provide – see Broder et. al. [5] for an example of the problems here). Since many of these plots require multiple (often thousands) of underlying "searches", the system has fairly rigorous performance requirements so as to be able to return the results to the user in seconds rather than hours.

Unstructured data requires the generation of metadata in documents to support general and task-specific queries. Most authors do not (and in many cases cannot) anticipate all of the tags that might be needed someday, and the size of the corpus makes manual tagging impossible. Robust automatic metadata annotation systems, such as UIMA [9, 11], are required to derive the higher-level concepts that need to be queried. In the example in Figure 1 (right), we are looking at "network prime time shows" mentioned on the web, even though that exact phrase probably does not occur. Instead, at ingestion time, an annotator examines each document and identifies shows that are being mentioned in any of a number of ways, and then annotates the document with metadata indicating which network the show is carried on.



Your Search of (ibm webfountain) returned the following results in 6.42 seconds.

1 - 10 of 558 results.
0 duplicates removed from a sample of 1,000
Approximate Enumeration Size: 5,300

Next Page

1. John Battelle's Searchblog: WebFountain, the Long Version
http://battellemedia.com/archives/000428.php
Enter **IBM** . » **IBM** s **WebFountain** review from Chichi Michi . **WebFountain** is a classic **IBM** solution to the search problem . **WebFountain** the Long Version . **IBM** PR response is interesting . With **WebFountain IBM** has sliced the web into subjective structured datasets . John Battelle has been to **IBM** Almaden and was given the grand tour of the home of **IBM Webfountain** . **[49 more]**
Date: May 1, 2006 *Rank*: 0.92577314

2. IBM sets out to make sense of the Web | CNET News.com
http://news.com.com/2100-1032-5153627.html
Salesforce.com sold on **IBM** Sybase technologies . With that insight **WebFountain** was born . **IBM** thinks Fast for bioscience searches . Trial over safety at **IBM** in jurors hands . By contrast **IBM** s **WebFountain** wants to help find meaning in the glut of online data . **[29 more]**
Date: Feb 5, 2004 *Rank*: 0.8710282

Figure 2: Enhanced search with additional result set statistics displayed in the sidebar.

Once the results are assembled, they need to be presented in a meaningful way. Such visualization also requires the ability to drill down to the supporting documents, as the underlying information may not be as accurate as is typical in more structured domains. Because of this, analysts find the ability to rapidly "spot check" certain results essential.

This is not to say that the system doesn't support keyword search queries. We've observed that most users begin to explore the system using keyword search, as it provides a familiar interface and a "feel" for the underlying data. Search therefore remains a core function of any unstructured management system and has to be done well.

Providing a reasonable level of abstraction between the applications and infrastructure requires a well defined API by which the applications can issue their questions and receive answers. This semi-structured query language allows traditional business intelligence style aggregation operators (e.g., COUNT, GROUPBY, UNION, INTERSECTION and their combinations) over lower level more search like subqueries such as ("IBM" or "I.B.M.") NEAR "DB2". Due to the complex nature of these queries we use a tree structure (i.e., XML) to represent queries rather than a left-to-right structure, but it otherwise the API strongly evidences its SQL roots.

2.2 People are not going to learn something new unless they have to

A user interface for discovering information buried within a large corpus of data poses challenges in itself, namely how the users interact with and drive the system. We have found that our clients are not naive users, thus a certain amount of analytic expertise and sophistication can be assumed. Nevertheless, a successful solution needs to be simple enough to provide a shallow learning curve, but powerful enough to allow experienced users to achieve maximum benefit. We are guided by the following principles:

- Don't overly complicate the system – it should be simple to start using the tools, even though the tools themselves may be quite complex. We shouldn't require the users to thoroughly understand everything before they can start achieving results. To quote Alan Kay, “Simple things should be simple, complex things should be possible.”

- Employ paradigms users are familiar with – if the users just want to do a simple search, let them. Use familiar visual cues, such as search bars, graphs and navigation (links) between tools. It doesn’t matter if a “better” approach is twice as fast if it takes years to learn how to use it.
- Interconnect as much as possible – relate functions together so that users can move between different tools/views of data. We provide applications that go beyond search. Combining search with other functions puts users in a familiar setting while augmenting the results with more detail, such as aggregate data. People understand search, but visualizing more information than just a result list can provide a fuller picture of the underlying data. For example, searching on a keyword returns a page showing not only a subset of matched documents, but also a distribution of the top `k` instances of various entities as found in the entire result set, e.g. names, email addresses, and dates. Users can then drill down or filter on any of these entities. See Figure 2 for an example of such a system.
- Grasping capabilities is not easy – users can start with search, and through it learn to harness more power of the analytics. However, this is a system for analysts – actually understanding the big picture in the data returned requires the experience to separate the valuable details from the chaff.
- Give users the power to narrow the corpus – allowing users to run analytics over a smaller set of documents lets them work within a manageable problem domain and reduces noise from irrelevant data. For example, users can reduce the scope by collections of sites; sets of entities such as documents containing names, emails or other sets; and other queries. This is especially useful when the users can create source collections of “authoritative” sources for a particular domain. Depending on the query, authoritative might be the MTV web site, or the American Heart Association – but few queries benefit from including both.

2.3 Sample early, sample often, and leave everything to chance

The benefits of sampling in traditional structured databases in the context of business intelligence type queries is well known. We have found that similar benefits can be derived from equivalent approaches in the unstructured content management domains that our clients are interested in.

If one has over 2 petabytes of data, streaming the entire corpus off disk and through the processors can take several days, even for a fairly large cluster. This means that for real-time query processing some kind of sampling is a must, especially for business intelligence style applications. Fortunately, statistical sampling provides significant performance advantages over traditional full-scan systems while still delivering highly accurate results.

Making a virtue out of this requirement, we have found that embracing random distribution simplifies and enhances many aspects of our platform. Documents are randomly distributed across the cluster by hashing a primary key (e.g. URL) to a cryptographically random 128 bit Unique Entity Identifier (UEID) and using the low order bits of this UEID to determine which node of the cluster to assign the document to.

This random assignment allows for the even distribution of documents across the cluster with minimal overhead, and makes it easy to determine where any given document resides on the cluster. A uniform and random distribution can be assumed, enabling analytics to pursue several approaches for random sampling.

In addition, the index on each node returns documents in UEID order (that is, fixed but random). Thus any sample out of an index is an unbiased sample of all the data on that node, which is an unbiased sample of all the data on the cluster. This allows queries such as “how many documents mention IBM in the same sentence as a database” to be processed through an estimator that converges like a standard polling sample. Typical 99% confidence level on a 50% true response from the entire population would have a margin of error:

$$MoE \approx 1.29/\sqrt{2n}$$

The factor of 2 is due to the index only returning positive hits (the negatives being the documents the index doesn't return). So if 10,000 samples are pulled (even off just a single node), the margin of error is 0.91% – probably good enough for most purposes. It should be noted that this serves as an upper bound. The actual MoE converges faster if the sample represents more or less than half the documents. Larger samples can easily be drawn nearly as quickly by consulting multiple nodes, but the MoE decreases non-linearly due to the square root. Unsurprisingly, it is much faster to pull 10,000 examples than it is to pull the entire 6 billion documents.

One example application which takes advantage of this sampling in our system allows users to search for the top websites (as opposed to web pages) discussing a specific query topic. Without storing website information directly in postings lists, this type of query would take prohibitively long without random sampling: it would be necessary to look up the hostname for each document hit, and aggregate across all matches. Random sampling allows us to examine a relatively small portion of the entire corpus, while retaining confidence in the accuracy of the result set.

New approaches to sample from a sorted index have recently been published in [5, 6]. However, the number of disk seeks required by these solutions is still orders of magnitude higher compared to using an index that already provides a uniform and random distribution. Most discovery applications rely on sampling to provide summaries of the underlying data. As such, we have found that an infrastructure that is optimized for these use cases is beneficial.

3 Hardware Observations

3.1 I/O is *the* bottleneck

With very large amounts of data it is almost always the case that the data an application is looking for is not in local memory – it either resides on local disk, or on a machine somewhere else on the local network. Any I/O operation goes through multiple layers: application call in user space, the file system or TCP stack in the kernel, and the network card or disk system in reality. Not surprisingly, with all these steps, there are very few analytics that do not end up bounded at some point by the amount of available I/O bandwidth.

For the disk subsystem, a sequential read performance on the order of 100 MB/sec is typical when using enterprise class drives and a RAID configuration. However, with discovery applications sequential access is rare. Much more common is randomly accessing many small files (on the order of 30kB each). Due to inefficiencies in every level of the I/O stack, a more realistic performance point for these smaller data units is 1-10 MB/sec, a drop of one or two orders of magnitude. This "narrow straw" to the data means that for semantic super computing systems handling billions of text documents, performance is often more limited by slow I/O than by lack of processing power.

One solution is to increase the number of "straws". This is one of the pressures that drives semantic super computing systems to scale out to large numbers of commodity systems: each provides one such straw. Taken together, a 256-node system can begin to process data at rates that allow reasonable performance at billions.

3.2 Newton's Law trumps Moore's Law

Why are the disk seek times so poor? Ultimately the culprit is that disk seek times have not shrunk nearly as fast as CPU speed has grown, because there are fundamental limitations on how fast a disk head can be accelerated and decelerated. This means that the time it takes to seek to a location on a disk and read a document (in this case around 30kB) quickly becomes a limiting factor. Making this even worse is that disk sizes are rapidly growing, thus leading to a tendency to reduce the number of disks that can be working in parallel.

Techniques like Asynchronous I/O (AIO) and Tagged Queuing emerging on SATA drives have helped, but ultimately some classes of problems are limited by this underlying seek time. This means that the most meaningful way to improve performance are optimizations throughout the I/O stack to limit seek counts. Additional strategies are to get more disks working on the problem in parallel through further distribution of the RAID arrays, optimization of allocation strategy, and the optimization of directory structure balancing algorithms.

Even with the directory tree traversal requiring multiple seeks, astonishing performance for both random reads and writes for file systems with up to one million files has been achieved. However, as one moves to billions of files, the overhead of managing thousands of separate file systems becomes prohibitive. We hope that in the near future, commercial file systems will be able to scale comfortably to over a billion files.

Until then, distribution of data across multiple file systems is the only solution. Taking advantage of distributed solutions leads to the question of how many nodes are required. Even at 256 nodes, each node needs to manage roughly four million files to reach a billion. At first glance, this seems to be well within the theoretical limit for the most commonly available file system implementations. In practice, as available space for allocation shrinks, a whole set of factors such as disk response time, RAID overhead, directory tree size and file system fragmentation can have a negative effect. The performance implications of storing well over 20 million files into a file system are significantly worse than initially anticipated. Overall performance deteriorates by more than 60% on average.

Some file system implementations deteriorate more rapidly than others. However, the trend is shared among all examined file systems, with significant performance degradation at volumes lower than 50% of their maximum theoretical limits. The good news is that off-the-shelf file systems can cope with millions of files in a single file system if the directory layout is carefully designed. The random reads required by discovery applications are more than an order of magnitude more costly than sequential reads. This single consideration seems to dominate effective read performance within our experiments. Intuitively, one would think that smaller directories with a three- or four-level deep directory structure would be optimal for such file systems as ReiserFS or XFS. However, this is not the case, as they often perform better when the millions of files are stored in a single directory. The results of our experiments suggest that systems need to be developed that perform well in all variations of the Create, Read, Update, Delete (CRUD) process, even when scaled to billions of files.

4 Operational Aspects

4.1 Cluster management by log file doesn’t work

Like a high-performance race car, any large-scale computing environment requires a solid base supported by constant tuning. Spend too much time sifting through mountains of system and application details and key trends, alerts, cycles and patterns are missed. Typical IT shops achieve a server-to-admin ratio of about 20:1. To improve that ratio by an order of magnitude, one must develop and deploy the proper toolset that provides efficiency and automation. In our environment, we leverage a variety of complementary technologies and products that enable us to achieve desired uptime and monitoring goals, as well as drastically reduce the number of administrators required. The suite of technologies employed ranges from a proprietary cluster management solution to freely available open source toolsets such as Nagios [3], MRTG [2], System Imager [4], etc.

We established Nagios as our central repository for the critical system and operational data, then used its robust alerting engine to build proper rules. Using Nagios’ SNMP capabilities, we configured our cluster to generate alerts on a variety of issues such as disk failures, RAID rebuild warnings, high environment temperatures, SSL certificate expiration, and more. Through the use of custom plug-ins, we extended the use of Nagios to monitor thousands of web services in our service oriented architecture.

MRTG’s graphing engine is used to graph any numerical output. We use MRTG to visualize all real-time performance and operational status, whether it be SNMP or application-generated data. As problems occur, this technique reduces the time spent performing costly root cause analyses, and also serves as a key quality control tool. The ability to get high-level summaries of trends and issues is critical as examining low-level alerts from thousands of machines on a daily basis is laboriously expensive.

System Imager excels at deploying images to identical or similarly-configured hardware configurations. Using a static DHCP and PXE-based boot process, cluster installations for 1000 servers can be performed in under 12 hours by a single person. It also makes for an effective disaster recovery tool. We deployed two enterprise management tools created by IBM Research to improve our operational efficiencies. IBM's Cluster Management System launches and manages the thousands of web services in our cluster, organizing services and machines into logical groupings that enabled one-click operations. We also implemented BlueBoard [15], a touch screen-based tool that utilizes variable-length, multi-dimensional "star" icons to represent system or application status. An operator can spend a few minutes reviewing the cluster operations using this interactive tool, with problem details and closure being only a few finger taps away.

Finding the right toolset for a cluster environment is key. By reducing the time administrators spend collecting and analyzing an environment's system and operational data, they will have more time to fine-tune your high-performance race car.

4.2 SLAs are odd

Negotiating service level agreements with hosting centers for a semantic super computing cluster differs from the traditional data center expectations, where upwards of 99% server availability is the rule. Hosting a very large cluster under such a simple performance metric is not particularly meaningful, as well as cost-prohibitive. This kind of machine-level availability is required if the ability to query the data hinges on every machine being responsive. In such cases, given:

- A node count of N
- A node availability rate of $P(SA)$
- An assumption that node failures are independent

the probability of query availability ($P(QA)$) can be expressed by

$$P(QA) = P(SA)^N$$

Assuming a 256-node cluster with one machine per node, to achieve a typical data availability target of 95% would require an expensive node availability target of 99.98%. Traditionally, this would often be implemented with redundant hardware, double-ended access to storage volumes, and other fairly expensive approaches.

Fortunately, due to sampling, our business intelligence style analytics do not require near-complete data availability to be able to support meaningful sampling and aggregation. As discussed in Section 2.3, we randomly distribute the data across the cluster, so sampling can be done on any available nodes without bias. For a given query, we merely need to identify how many nodes are needed to provide an estimate within the error bounds required of the sample.

We have found that for the kinds of questions our clients are interested in, solutions will perform well if at least 80% of the nodes are available. Thus, for our 256 node example we need:

$$P(QA) = \sum_{n=205}^{256} \binom{256}{n} P(SA)^n (1 - P(SA))^{(256-n)}$$

To achieve a data consistency service level of 95%, we only need a single-node service availability of 85% - a much more cost effective service point.

Why all this focus on SLAs and availability? With large systems, failures are not a risk but a certainty. Our super computing cluster experiences over 350 hard drive failures a year, 20 CPU failures, and numerous power supply failures, memory module failures, etc. The more flexibility available to address these problems, the lower the cost of doing so. While not quite "fail in place" [10], the 85% target provides us the ability to handle small and medium outages without degradation of data quality for our customers and their analytic applications.

5 Middleware Characteristics

5.1 Scatter/gather is non-trivial at 256 nodes and up

The promise of scale-out performance improvement is predicated on the ability to efficiently scatter queries to a cluster and gather the results back. Initially, seamless linear scaling can be achieved with simple solutions. As the degree of system scale-out increases, so does its complexity, both from hardware- and software-stack perspectives. Logically a client connects to a multiplexer component for a single entry point which hides the underlying large-scale complex system. Ideally, clients maintain a high degree of abstraction both at the logical and physical layers. The client should only be aware of very simple interaction parameters, which in this case are mostly time constants on query response. Reality though, is somewhat more complex. Consider what can go wrong when a query is sent out to 256 nodes:

- Some of the 256 nodes will almost certainly not be online (see Section 4.2).
- Of those online, a few might be either hung or responding very slowly due to hardware issues, machine load, periodic maintenance or upgrades, etc.
- The network switch or the multiplexer machine may become overwhelmed with traffic and start dropping or corrupting packets, run out of resources such as memory or socket descriptors.
- Mismatched network driver configurations may result in packet errors or performance degradation due to different MTU sizes or default duplex settings, which are not uniform in multi-vendor environments.
- Other issues include exceeding the maximum number of open file handles; inconsistencies within the system-wide configuration (kernel, memory, binaries, etc); and inconsistent access control configuration on runtime directories.

We have seen all of these problems and more. As a result, there is no such thing as a "maximum time" a query to the cluster may take, since at least one node will always time out. The multiplexer abstracts out this complexity, by hiding the detailed configuration and states of the system. It is required to handle error conditions, failures and partial failures of all related components. The multiplexer is also responsible for scattering the request to the individual components and gathering the results back. Each component executes independently, without any information about other nodes. Through the use of a highly optimized multiplexer, near-linear performance scaling has been achieved as the system scaled to the hundreds of nodes.

There is no single error-handling strategy that fulfills the needs of all clients. Instead, the state of each node is reported, so that the client can decide what the appropriate course of action is. The client can specify the mode of operation, which is used to tune the query execution.

6 Data Representation and Algorithmic Insights

6.1 One kilobyte per document over 6 billion documents is 6 terabytes

When dealing with very large numbers of documents it becomes necessary to pay a high degree of attention to the way storage trade-offs are handled that is quite foreign to today's programmers. Considerable thought must go into trading space for speed, data representations, and blob storage, or the system can quickly run out of space and/or get bogged down in I/O.

Semantic super computing systems constantly need to balance CPU usage against storage usage. Disk space can be preserved by compressing data and metadata or by generating metadata on demand – trading additional CPU overhead for less storage and I/O. Conversely, if the system has to respond as fast as possible, storing everything uncompressed may be the best solution. What to store and how to store it gains in importance, as the size of the corpus grows. One additional kilobyte per document over one billion documents requires an extra terabyte of storage. Furthermore, a larger corpus complicates the migration from one data format to another. Analyzing the storage footprint of an annotator and peer-reviewing the chosen data format becomes increasingly important.

Finding a good trade-off between CPU and storage can be tricky. We found that the best solutions are often counter-intuitive. In semantic applications that focus on unstructured textual information, standard compression using `gzip`[7] provides an excellent level of space savings at a reasonable performance cost. We apply `gzip` to the whole document and all metadata before committing it to disk. One surprising insight is that human-readable formats can outperform "hand tuned" binary formats after compression [13]. For example, representing an array of numbers as a string of white space separated values yields same or better data size after compression compared to a binary representation using BER encoding[1]. In addition, human readability clearly benefits developers as it simplifies debugging.

Another finding is that the use of Base64 encoding (a typical approach for placing "blobs" in XML) has surprisingly severe CPU and storage overheads. Some of this is obvious – Base64-encoded documents are roughly 30% larger than the original content, but there is a more insidious effect when compression is considered: the encoded content compresses quite poorly because a repeated sequence of bytes in the original document may be encoded in three different ways depending on the alignment in the data string, resulting in a potentially large decrease in compression efficiency.

Most data structures in our system contain large amounts of textual data, augmented with some binary data. One example is tokenized content, which contains each token (string), an offset (integer) and a length (integer). We found that converting the binary integer data to text results in better performance, lower storage requirements and easier debugging, than serializing such a data structure into a binary representation.

6.2 The need for speed, and 5% is only 1 part in 20

As is the case with any project that has a strong research component, there is always a desire to improve the algorithms used. But because we deal with such large volumes of data, this desire for increased accuracy must be balanced against the net performance impact when applying the algorithm across billions of documents.

Consider an algorithm that has been improved to produce results which are 5% more accurate than its predecessor. If one is given an infinite amount of time, and/or infinite computing resources, then the algorithm could be applied to a complete corpus and would result in an improvement in accuracy.

But in the absence of infinite time or computing resources, trade-offs need to be made. If the improved algorithm is quite a bit slower than the older algorithm, then it will process fewer samples in its allotted time. Fewer samples would likely produce overall *less* accurate results, even after taking into account the improved precision.

Due to the use of sampling for much of our work there is one aspect where it is almost always worthwhile to strive for improved algorithms: the removal of bias. Algorithms that introduce systematic bias or errors will produce bad results no matter how many samples are used.

7 Testing

7.1 When testing in the billions, "good enough" is not good enough

According to Murphy's law, whatever can go wrong will go wrong. This is especially applicable when designing a solution that is meant to process billions of documents. Consider the implementation of an efficient tokenizer, which needs to process thousands of pages per second. To avoid costly memory allocations, it may assume that no token is larger than 32 bytes, and pre-allocate memory accordingly. This implementation may work on a test run of a million documents, but will invariably encounter problems when applied to billion document corpora. For example, long concatenated nouns are common in German. Moreover, some data sources provide lower-quality data, such as Web pages with thousands of words and not a single white-space separator.

The above example may seem trivial, but when architecting a robust analytics platform that leverages in-house and third-party annotators [9, 11], and needs to survive whatever oddity it encounters, the only way to cope is by defensive design. No amount of testing can cover all scenarios that may eventually occur. The system must be able to gracefully handle crashing annotators without requiring manual intervention.

On the other hand, too much self-healing may degrade overall performance, necessitating the termination of a defective annotator. Unfortunately, most complex annotations depend on a chain of annotators that work in concert, and having one of them ejected can produce unpredictable results. Once again, there is no perfect solution. Just because a component was tested against a million documents, doesn't mean that it will survive the rigors of a billion pages.

8 Conclusions

In this paper, we presented our hard-won lessons for handling information storage, retrieval and analysis requirements at the multiple billion document level. We showed how, at this level, some of the traditional approaches to information management no longer apply, and how some of the best approaches may initially seem counter-intuitive. We hope that our experience proves useful for those designing their own large scale document systems.

References

[1] Basic encoding rules. `http://en.wikipedia.org/wiki/Basic_Encoding_Rules`.

[2] Mrtg. `http://www.mrtg.com`.

[3] Nagios. `http://www.nagios.org`.

[4] System imager. `http://www.systemimager.org`.

[5] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 245–256, New York, NY, USA, 2005. ACM Press.

[6] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 367–376, New York, NY, USA, 2006. ACM Press.

[7] P. Deutsch. Gzip file format specification version 4.3. RFC 1952, 1996.

[8] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW*, pages 178–186, 2003.

[9] D. Ferrucci and A. Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, 2004.

[10] C. Fleiner, R. B. Garner, J. L. Hafner, K. K. Rao, D. R. Kenchammana-Hosekote, W. W. Wilcke, and J. S. Glider. Reliability of modular mesh-connected intelligent storage brick systems. *IBM Journal of Research and Development*, 50(2-3):199–208, 2006.

[11] T. Götz and O. Suhre. Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43(3):476–489, 2004.

[12] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[13] D. Gruhl, D. N. Meredith, and J. Pieper. A case study on alternate representations of data structures in xml. In A. Wiley and P. R. King, editors, *ACM Symposium on Document Engineering*, pages 217–219. ACM, 2005.

[14] D. Gruhl, D. N. Meredith, J. Pieper, A. Cozzi, and S. Dill. The web beyond popularity: a really simple system for web scale rss. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 183–192. ACM, 2006.

[15] D. M. Russell, A. Dieberger, V. Bhagwan, and D. Gruhl. Large visualizations for system monitoring of complex, heterogeneous systems. In *INTERACT*, pages 938–941, 2005.

Web Information Extraction and User Modeling: Towards Closing the Gap

Eugene Agichtein
Mathematics and Computer Science Department
Emory University
eugene@mathcs.emory.edu

Abstract

Web search engines have become the primary method of accessing information on the web. Billions of queries are submitted to major web search engines, reflecting a wide range of information needs. While significant progress has been made on improving the relevance of the results, web search process often remains a frustrating experience. At the same time, web information extraction has seen tremendous progress, such that knowledge bases of millions of facts extracted from the web are now a reality. Yet it is not clear how effectively these knowledge bases support common user information needs. We posit that a key for web information extraction to significantly impact the web search experience is to connect the extraction process with user modeling, particularly with automatic methods for inferring user information needs and anticipated interaction patterns. In this paper we overview some recent efforts for user modeling and inferring user preferences in the context of closing the gap between web information extraction and user modeling.

1 Introduction

The ultimate goal of web search is to provide answers for user information needs – where the answers may be documents, lists of items for sale, lists of structured objects, or even multi-document summaries. According to recent studies, between 39% and 60% of the queries submitted to web search engines are informational in nature [9]. The information needs behind the queries have been specifically classified into Directed, Undirected, Advice, Locate, and List categories. Of these, the Directed, Undirected, and Locate categories account for more than 56% of all queries [35]. Many of these information needs may be answered more effectively by providing structured information –by allowing precise queries, and integrating and aggregating relevant information from multiple documents.

Significant progress has been made on scalable and accurate web information extraction, allowing state-of-the-art systems to automatically extract knowledge bases of millions of facts from hundreds of millions of web pages (e.g., [17, 31]), thus bridging the gap between structured and unstructured data. These efforts have focused on accuracy and scalability of information extraction. Yet, a different gap –between user modeling to infer user information needs and the information extraction process– remains largely open. Recently, user behavior has been shown to be one of the most valuable indicators for accurate web search (e.g., [1]), and is becoming increasingly useful as more powerful and unobtrusive monitoring and analysis techniques are developed. The

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

question we explore in this paper is how user behavior –via user modeling– might be useful to guide and tune web information extraction. We describe the building blocks towards bridging the gap between web information extraction (Section 2) and user modeling (Section 3), and outline related research questions that we currently pursuing (Section 4).

2 Web Information Extraction: User Control vs. Effort

We briefly survey web information extraction from the user perspective –focusing on the experience of a casual web searcher rather than a developer or sophisticated analyst. Information extraction refers to the process of representing information in text in a structured form, and representative tasks include identifying entities, facts, events and relations extracted from the documents in either the surface web [28] or in the "deep web" [32]. We briefly review different types of extraction systems, categorized by the way the user input is solicited, and how it is incorporated into the extraction process. This is not a comprehensive review, and we only mention a few representative systems among the many influential information extraction systems reported.

Language Engineering: This approach to information extraction has stood the test of time as the resulting systems performed consistently well in DARPA-sponsored complex information extraction scenarios such as terrorist attacks, natural disasters, and corporate succession events extracted mostly from newspaper text. These systems offer rich development environments for constructing, manipulating, and testing extraction rules and lexicons for a variety of information extraction steps. With sufficient domain knowledge, resources, and expert tuning, this approach can result in highly accurate systems. A prominent example is the *PET* system [38], developed by the Proteus project at the New York University. *PET* allows a system user (typically a domain expert or developer) to create, generalize, and test extraction patterns based on the manual examination of example text documents. Another example of a language-engineering environment is *GATE* [15], a system developed at the University of Sheffield. In both systems, the actual users of the final extraction system either need to acquire the expertise with the engineering environment (an unlikely prospect), or more commonly must interact with the developers to build a system that supports the users' information needs. Typically this is an iterative process that may require weeks or months depending on the complexity of the task.

Supervised Machine Learning: A promising approach is to automatically train an information extraction system and generate rules or extraction patterns for new tasks. The training is done over a large *manually tagged* corpus, where a system can apply machine learning techniques to generate extraction patterns. Examples of such systems include CRYSTAL [20], BWI [25], and Rapier [11], and successful extraction tasks include address segmentation, entity tagging, resume field extraction, gene and protein interactions and many others. A drawback of this approach is the need for a large tagged corpus, which involves a significant amount of manual labor to create. To reduce the amount of required annotation, the AutoSlug system [33] generated extraction patterns automatically by using a training corpus of documents labeled as either relevant or irrelevant for the topic. This requires less manual labor than annotating the documents, but nevertheless the effort involved is substantial. In all cases above, a large annotated corpus was created - a daunting prospect for most casual users. In this setting, the user "interacts" with the system by defining the target entity types and relations or templates, and providing many tagged training instances. The user needs are then "frozen" in the beginning of the training process. Unfortunately, as user needs evolve, or the text corpus changes, the labeling process would have to be re-done, potentially from scratch.

Partially Supervised Machine Learning: A powerful approach to exploiting large amounts of *un-annotated* text was proposed in [14, 39], where starting with a few *seed extraction patterns*, a system can acquire and refine additional patterns. In this setting, the user may either provide patterns directly, select and possibly extend some subset of patterns in the system, or interact with the environment (e.g., *Proteus*) to build patterns. A significant improvement to the approach was introduced by the KnowItAll system [17] and subsequent variants, which starts with *general* patterns (shared by all extraction tasks) and proceeds to automatically refine and generate specific rules for particular classes of entities or relationships. A different approach was recently developed in the ODIE project [23, 36] for automatically clustering and classifying co-occurring pairs of entities given a description of the topic of interest. In this setting, the user input is limited to a description (e.g., a few keywords) for a topic, and the system attempts to discover important relations and entities. More recently, the URES system [19] was introduced that operates similarly, but focuses on the rare tuples by generating more flexible extraction patterns (e.g., with gaps). These approaches are –by design– almost completely unsupervised, or data-driven, and hence the user has minimal participation in the extraction process per se. Rather, a user may provide restrictions at query time by specifying the type(s) of entities or relations to use to process the query.

A complementary approach for extracting relations was introduced by DIPRE [8] and significantly extended in the *Snowball* system [5, 6]. *Snowball* starts with a few user-provided examples, or *seed tuples*, for the target relation. To identify new tuples, the named entities of interest must be embedded in similar contexts as the seed tuples. *Snowball* was designed to be flexible about variations inherent in natural language text and can in some cases discover a vast majority of the tuples in a collection while starting with just a few example tuples. This variant of partially-supervised extraction allows a user to focus on particular subset of tuples by manipulating the seed tuples. Nevertheless, both pattern-seeded and tuple-seeded approaches assume a "batch" mode of interaction (i.e., that the same relations will be used in the future) and is not amenable to "one-of" question answering, or on-demand extraction; Also, such systems may have to be re-tuned (albeit at much lower costs) for new tasks or with changes in the user information needs.

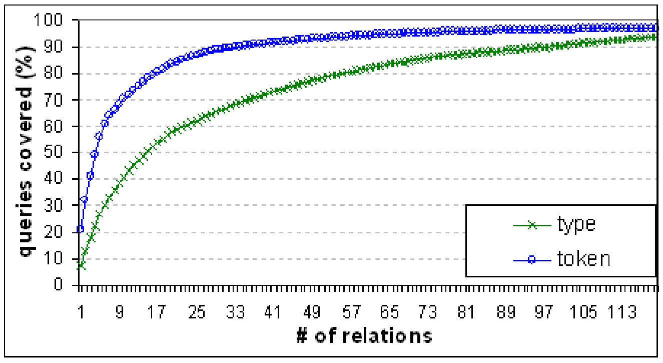
Web Question Answering: The task of returning short answers to natural language questions – where the user information need is inferred at query time is commonly referred to as question answering (QA). Many question answering systems are represented in the yearly TREC Question Answering competition (e.g., [37]). For example, Moldovan et al., and Aliod et al. [29, 3] present systems that re-rank and postprocess the results of regular information retrieval systems with the goal of returning the best passages. Cardie et al. [12] describe a system that combines statistical and linguistic knowledge for question answering and employs sophisticated linguistic filters to postprocess the retrieved documents and extract the most promising passages to answer a question. In all cases, the user (in principle) can specify any information need over supported answer types at query time. In practice, the questions usually are simple "factoid" question – that could be extracted from a short string in a single document, and thus a QA system typically cannot support aggregate queries, range queries and joins that would be possible if the information was pre-computed by an information extraction system. In order to pre-extract the necessary information to answer such queries, good understanding of the user information needs and the anticipated interactions and access patterns is required, as we discuss next.

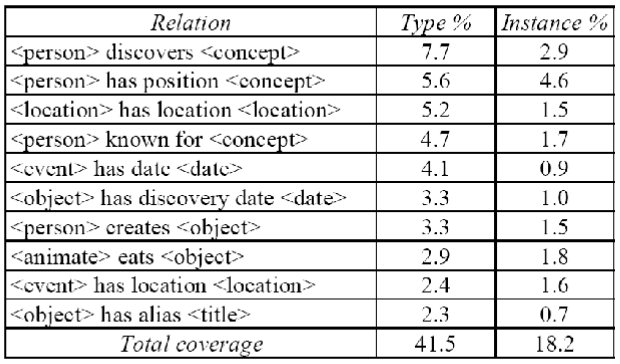
3 User Modeling: What Do Users Want?

Understanding user information needs and query intents are crucial tasks for accurate information retrieval and web search. As information needs vary widely, user modeling research evolved largely along two paths: careful manual analysis of the queries and the information needs behind them, as well as *automatically* inferring information need through models of observable actions of varying complexity (e.g., query refinements, result interactions, and browsing patterns). Again, this section is not meant to be a comprehensive overview of user modeling (please refer to excellent books on the subject, e.g., [27]). Rather, we focus on techniques we developed recently that may be helpful for improving web information extraction performance.

3.1 Manual Analysis of User Information Needs

Among many domains that have benefited from information extraction, some of the most extensive user and query modeling and analysis has been done for factoid question answering [37], and an important variant of the task, medical question answering (e.g., [26]). Other domains include shopping, image and library search (e.g., faceted search and browsing [16]) for which distinct user models have been developed based on extensive manual analysis. In the TREC domain, fine-grained question types have been developed, such as the 140 question-type Webclopedia question taxonomy [21], with associated rules for mapping questions to answer types. As another related approach, [22] used a large number of dictionaries, or lists, some of which were constructed dynamically by querying sites such as Amazon.com based on careful analysis of previous, and anticipated, TREC QA queries.





Relation	*Type %*	*Instance %*
<person> discovers <concept>	7.7	2.9
<person> has position <concept>	5.6	4.6
<location> has location <location>	5.2	1.5
<person> known for <concept>	4.7	1.7
<event> has date <date>	4.1	0.9
<object> has discovery date <date>	3.3	1.0
<person> creates <object>	3.3	1.5
<animate> eats <object>	2.9	1.8
<event> has location <location>	2.4	1.6
<object> has alias <title>	2.3	0.7
Total coverage	41.5	18.2

Figure 1: Coverage of queries by type and token (a) and the list of the most frequently queried relation types (b) [4].

.

In the web search domain, questions do not necessarily follow the TREC model as the TREC questions are grammatical, and well-formed, whereas the web questions are often short, and vague. To better understand the user needs expressed in web search, we performed an extensive examination of more than 2,000 question-like queries sampled from the millions of queries submitted to the web-searchable Microsoft Encarta. Specifically, we focused on questions that could be answered by some binary (and possibly tertiary) relation. Interestingly, a fairly small set of relations cover a large fraction of user queries. Figure 1 (a) reports the number of relations (horizontal axes) that results in the reported coverage as a percentage of factoid questions in the sample (vertical axes). For illustration, Figure 1 (b) lists the relations identified by our annotators as most frequently needed to answer the questions in query log. From the sample, fewer than 25 relations are needed to cover more than 50% of the queries, with a very skewed frequency distribution of both relation types and relation instances. Reference [4] is a promising step towards answering factoid questions using knowledge bases extracted from trusted resources or the web. The skewed distribution of relationships observed in the annotated queries indicates that a limited number of fact tables can cover the bulk of user factoid questions. This approach could be extended to handling complex questions through decomposition into simpler factoid questions. Our results suggest focusing computational and annotation resources on extracting fact tables for frequently queried relationships, and on mapping user questions to appropriate relations. While for particularly important resources or questions types we could manually analyze query logs for re-tuning extraction, a more promising approach would be to *automatically* identify user interests, preferences, and access strategies from observable behavior data as we describe next.

3.2 Automatic Analysis of User Behavior to Infer Information Needs and Preferences

The best indicator of user intent and interest and relevance of results is explicit human feedback. Unfortunately, such feedback is expensive, and often not realistic to obtain. Hence, reducing the dependence on explicit human judgments by using implicit relevance feedback has been an active topic of research. Several research groups have evaluated the relationship between implicit measures and user interest. For example, reference [30] presented a framework for characterizing observable user behaviors using two dimensions-the underlying purpose of the observed behavior and the scope of the item being acted upon. Reference [10] studied how several implicit measures related to the interests of the user using a custom browser to gather data about implicit interest indicators and to probe for explicit judgments of Web pages visited. Reference [10] also found that indicators such as the time spent on a page, and the amount of scrolling on a page have a strong positive relationship with explicit interest.

In the context of web search, Fox et al. [18] explored the relationship between implicit and explicit measures. Reference [18] built an instrumented browser to collect data and developed Bayesian models to relate implicit measures and explicit relevance judgments for both individual queries and search sessions. They found that clickthrough was the most important individual variable but that predictive accuracy could be improved by using additional variables. Joachims et al. [24] presented an empirical evaluation of interpreting clickthrough evidence. By performing eye tracking studies and correlating predictions of their strategies with explicit ratings, the authors showed that it is possible to accurately interpret clickthrough events in a controlled, laboratory setting. More recently, Radlinski and Joachims [34] exploited session-level implicit feedback to improve ranking further.

Recently, we presented a real-world study of modeling the behavior of web search users to predict search result preferences [2]. We introduced more robust probabilistic techniques for interpreting clickthrough evidence by aggregating across users and queries, resulting in more accurate than previously published results on interpreting implicit feedback. Specifically, we focused on the *deviations* of user behavior from the “expected” behavior –estimated from the aggregated behavior patterns computed across all users and queries. Figure 2 illustrates this idea for one particular indicator, result clickthrough frequency. Figure 2(a) reports the overall clickthrough frequency, while Figure 2(b) separates the queries by the position of known top relevant result (PTR), and subtracts the “expected” clickthrough fraction at each corresponding position –thereby computing the deviation from the “expected” clickthrough at that position. For example, on average clickthrough at position 2 may be 0.55, but for queries with first *relevant* result at 2, clickthrough at position 2 is 0.65 for positive deviation of 0.1. By modeling these deviations, we can significantly increase the accuracy of clickthrough interpretation [2].

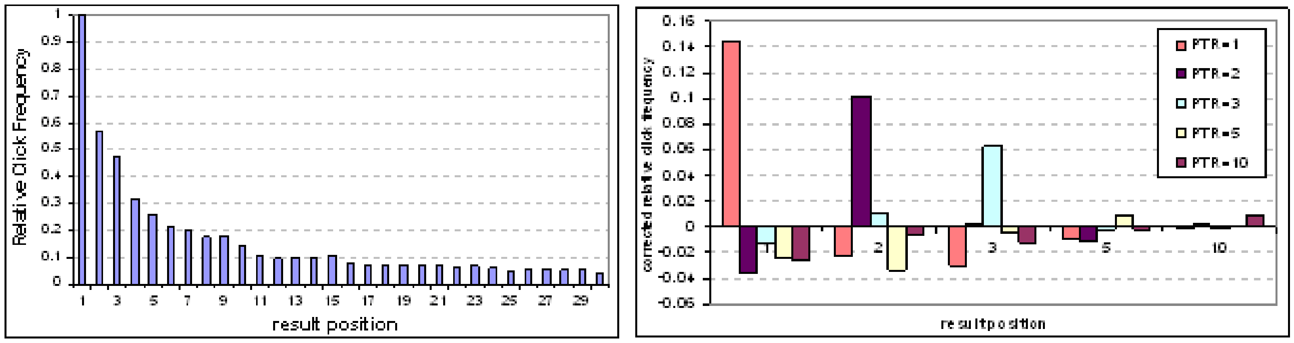
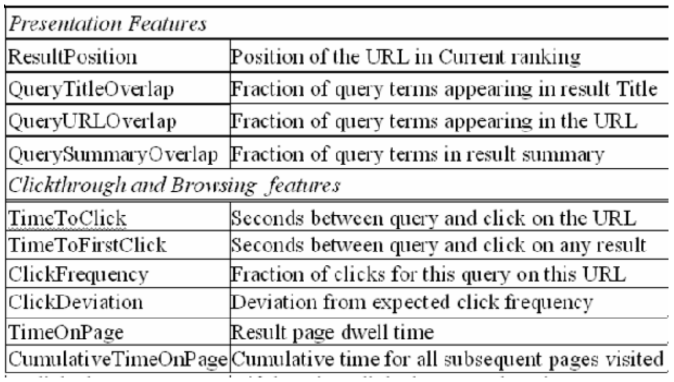


Figure 2: Overall clickthrough rate (a) and (b) Deviations from expected clickthrough rates for varying positions of top relevant result, per [2].

.

Furthermore, reference [2] introduced a general model for interpreting post-search user behavior that incorporates clickthrough, browsing, and presentation features. A sample of features, derived from server and client-side instrumentation, are shown in Figure 3(a). By considering the complete search experience after the initial query and click, we demonstrated prediction accuracy dramatically exceeding that of interpreting only the limited clickthrough information. Automatically learning to interpret user behavior results in substantially better performance than the human-designed ad-hoc clickthrough interpretation strategies, as reported in the pairwise agreement plot in Figure 3(b). The horizontal axes refers to the fraction of pairwise preferences (explicitly stated) that a system was able to predict correctly, and the vertical axes measures the fraction of preferences for which a prediction was attempted, that were predicted correctly. The full model, reported as *UserBehavior*, significantly outperforms the clickthrough Deviation model, which in turn outperforms the previous state-of-the-art clickthrough model.



Presentation Features	
ResultPosition	Position of the URL in Current ranking
QueryTitleOverlap	Fraction of query terms appearing in result Title
QueryURLOverlap	Fraction of query terms appearing in the URL
QuerySummaryOverlap	Fraction of query terms in result summary
Clickthrough and Browsing features	
TimeToClick	Seconds between query and click on the URL
TimeToFirstClick	Seconds between query and click on any result
ClickFrequency	Fraction of clicks for this query on this URL
ClickDeviation	Deviation from expected click frequency
TimeOnPage	Result page dwell time
CumulativeTimeOnPage	Cumulative time for all subsequent pages visited

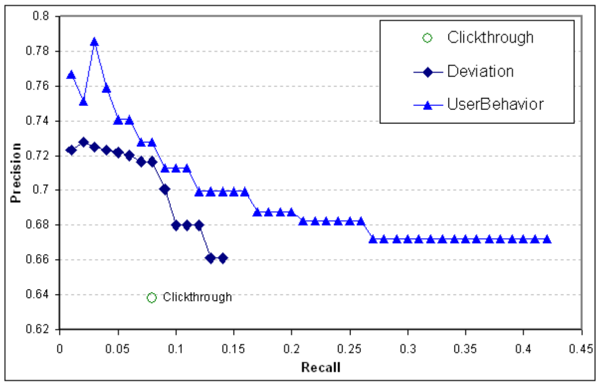


Figure 3: Sample user behavior features (a) and the accuracy of predicting user pairwise result preferences (b), per [2].

While our techniques perform well on average, our assumptions about clickthrough distributions (and learning the user behavior models) may not hold equally well for all queries. For example, queries with divergent access patterns (e.g., for ambiguous queries with multiple meanings) may result in behavior inconsistent with the model learned for all queries. Specifically, for informational queries, the user models and behavior patterns are expected to be divergent from navigational queries. In particular, we showed that we can distinguish navigational queries (and their target websites) from the rest of the queries by automatically classifying user behavior patterns [7]. The classification framework we developed is amenable to easy maintenance and updating, in particular to be tuned for evolving user behavior patterns and query distributions. A promising direction is how to extend and apply these models to identify implicitly structured queries (i.e., the queries that would benefit from information extraction) as such queries appear to also have distinct interaction patterns. This is one step towards integrating information extraction and automatic user modeling, as we discuss next.

4 Towards Integrating Web Information Extraction and User Modeling

So far we have considered some of the recent information extraction and user modeling research. If we knew the expected workload of types of relations to expect, we could more effectively optimize the resource allocation for the information extraction process accordingly. Unfortunately, a large gap remains between detailed manual analysis directly usable for extraction tuning (e.g., [4]) and the more course-grained automatic user modeling (e.g., [2]). The missing piece is the ability to automatically analyze the query and interactions stream, and to automatically identify the relations and entities that would have been helpful if pre-extracted for processing these queries. An interesting approach to improving question answering accuracy by automatically indexing the more frequent answer types was introduced recently by Chakrabarti et al. [13]. The distribution of entity types to be indexed is computed according to workload distribution of questions and answers from the TREC QA benchmark [37]. For the types of questions well represented in the TREC benchmark, this strategy is feasible. Unfortunately, web search queries exhibit different properties from the TREC questions, and hence tuning on the TREC set exclusively may not be optimal for the web search setting.

The gap between automatic user modeling and information extraction suggests a promising area of research that requires addressing multiple challenges. First, we need to identify queries amenable to answers from information extraction output. To do this, we can train a classifier (e.g., as in reference [7]) to identify queries and behavior patterns indicative of a user searching for a specific answer that could be pre-computed by information extraction. After such queries are identified, the next step is to infer the actual information need (and not just the documents likely to be relevant) – in effect to identify the types of answers appropriate for the query.

We are currently improving techniques for mining the user behavior information to improve the "resolution" of automatically identifying user information needs. Finally, the ranking of candidate answers generated from the extracted information could be improved by incorporating intelligent user modeling techniques and past user behavior.

To summarize, there is a large potential in "closing the loop" between web information extraction and automatic user modeling. However, so far the user need analysis for information extraction has been primarily manual. At the same time, automatic user modeling to infer information needs and preferences have so far focused on general web search, due to availability of both user data and explicit ratings. We described some of our recent results in user modeling –manual and automatic– that could potentially serve as first steps towards automatically tuning web information extraction systems, thus closing the gap between the information extraction process and user information needs.

ACKNOWLEDGEMENTS: The experimental results were reported in prior work ([4, 2, 7]) done at Microsoft Research in collaboration with Eric Brill, Silviu Cucerzan, Susan Dumais, and Zijian Zheng.

References

[1] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.

[2] Eugene Agichtein, Eric Brill, Susan Dumais, and Robert Ragno. Learning user interaction models for predicting web search result preferences. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.

[3] Diego Aliod, Jawad Berri, and Michael Hess. A real world implementation of answer extraction. In *Proceedings of the 9th International Workshop on Database and Expert Systems, Workshop: Natural Language and Information Systems*, 1998.

[4] Eugene Agichtein, Silviu Cucerzan, and Eric Brill. Analysis of factoid questions for effective relation extraction. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval Poster Session*, 2005.

[5] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, June 2000.

[6] Eugene Agichtein. *Extracting Relations From Large Text Collections*. PhD thesis, Columbia University, 2005.

[7] Eugene Agichtein and Zijian Zheng. Identifying "best bet" web search results by mining past user behavior. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[8] Sergey Brin. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB'98)*, March 1998.

[9] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 2002.

[10] Mark Claypool, David Brown, Phong Le, and Makoto Waseda. Inferring user interest. *IEEE Internet Computing*, 2001.

[11] Mary E. Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Sixteenth National Conference on Artificial Intelligence*, 1999.

[12] Claire Cardie, Vincent Ng, David Pierce, and Chris Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In *Proceedings of ANLP*, 2000.

[13] Soumen Chakrabarti, Kriti Puniyani, and Sujatha Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *Proceedings of the 15th International Conference on World Wide Web (WWW)*, 2006.

[14] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[15] Hamish Cunningham. *Software Architecture for Language Engineering*. PhD thesis, University of Sheffield, 2000.

[16] Wisam Dakka, Panagiotis G. Ipeirotis, and Kenneth R. Wood. Automatic construction of multifaceted browsing interfaces. In *Proceedings of the International Conference on Knowledge Management (CIKM)*, 2005.

[17] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *Proceedings of the World Wide Web Conference*, 2004.

[18] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems*, 2005.

[19] Ronen Feldman and Benjamin Rosenfeld. Boosting unsupervised relation extraction by using ner. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2006.

[20] David Fisher, Stephen Soderland, Joseph McCarthy, Fangfang Feng, and Wendy Lehnert. Description of the UMass systems as used for MUC-6. In *Proceedings of the 6th Message Understanding Conference*, 1995.

[21] H. Hovy, U. Hermjakob, and D. Ravichandran. A question/answer typology with surface text patterns. In *Proceedings of HLT*, 2001.

[22] Wesley Hildebrandt, Boris Katz, and Jimmy Lin. Answering definition questions with multiple knowledge sources. In *Proceedings of HLT/NAACL 2004*, 2004.

[23] Takaaki Hasegawa, Satoshi Sekine, and Ralph Grishman. Discovering relations among named entities from large corpora. In *Proceedings of the Annual Meeting of Association of Computational Linguistics (ACL)*, 2004.

[24] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2005.

[25] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[26] Jimmy Lin and Dina Demner-Fushman. The role of knowledge in conceptual retrieval: A study in the domain of clinical medicine. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.

[27] Gary Marchionini. *Information Seeking in Electronic Environments*. Cambridge University Press, 1997.

[28] Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *ACM Queue*, 3, 2005.

[29] Dan Moldovan, Sanda Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. Lasso: A tool for surfing the answer net. In *Proceedings of TREC-8*, 1999.

[30] D. Oard and J. Kim. Modeling information content using observable behavior. In *Proceedings of the 64 Annual Meeting of the American Society for Information Science and Technology*, 2001.

[31] Marius Pasca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: The one-million fact extraction challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[32] Sriram Raghavan and Héctor García-Molina. Crawling the hidden web. In *Proceedings of the Conference on Very Large Databases*, 2001.

[33] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[34] Filip Radlinski and Thorsten Joachims. Query chains: learning to rank from implicit feedback. In *Proceeding of the 11th ACM SIGKDD international Conference on Knowledge Discovery in Data Mining*, 2005.

[35] Daniel E. Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the International World Wide Web Conference (WWW)*, 2004.

[36] Satoshi Sekine. On-demand information extraction. In *Proceedings of the COLING/ACL 2006 Poster Session*, 2006.

[37] Ellen M. Voorhees. Overview of the TREC 2003 question answering track. In *Proceedings of the Text REtrieval Conference*, 2004.

[38] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[39] Roman Yangarber, Ralph Grishman, Pasi Tapanainen, and Silja Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of Conference on Applied Natural Language Processing (ANLP-NAACL)*, 2000.

Structured Queries Over Web Text

Michael J. Cafarella, Oren Etzioni, Dan Suciu
University of Washington
Seattle, WA 98195
{mjc, etzioni, suciu}@cs.washington.edu

Abstract

The Web contains a vast amount of text that can only be queried using simple keywords-in, documents-out *search queries. But Web text often contains structured elements, such as hotel location and price pairs embedded in a set of hotel reviews. Queries that process these structural text elements would be much more powerful than our current document-centric queries. Of course, text does not contain metadata or a schema, making it unclear what a structured text query means precisely. In this paper we describe three possible models for structured queries over text, each of which implies different query semantics and user interaction.*

1 Introduction

The Web contains a vast amount of data, most of it in the form of unstructured text. Search engines are the standard tools for querying this text, and generally perform just one type of query. In response to a few keywords, a search engine will return a relevance-ranked list of documents. Unfortunately, treating Web text as nothing but a collection of standalone documents ignores a substantial amount of embedded structure that is obvious to every human reader, even if current search engines are blind to it. Web text often has a rich, though implicit, structure that deserves a correspondingly-rich set of query tools.

For example, consider a website that contains hotel reviews. Although each review is a self-contained text that is authored by a single person, the set of all such reviews shows remarkable regularity in the type of information that is contained. Most reviews will contain standard values such as the hotel's price and general quality. Reviews of urban hotels might discuss how central the hotel's location is, and reviews of resorts will mention the quality of the beach and pool. In other words, the hotel reviews have a messy and implicit "schema" that is not designed by any database administrator but is nonetheless present.

If a query system made this structure explicit, users could navigate the hotel reviews using, say, a fragment of SQL. A user could easily list all hotels that are both very quiet and in central Manhattan, even though doing so would be impossible with a standard search engine.

Note that we are *not* suggesting that the hotel site publish any sort of structural metadata. We do not ask Web publishers to add any additional information about their text. Our proposed query systems preserve the near-universal applicability of search engines, using only structural elements that are *already present* in the text

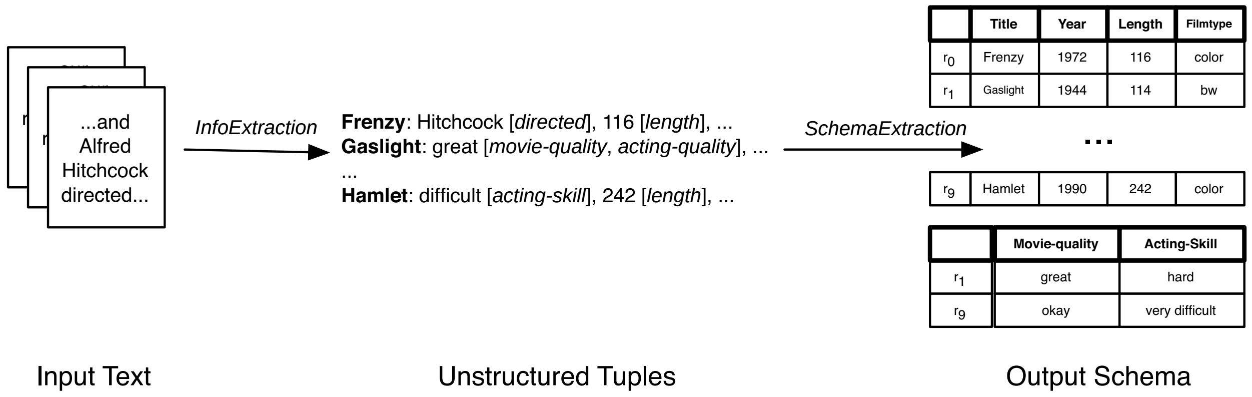


Figure 1: In the Schema Extraction Model, the query system receives some **Input Text** and runs it through an information extraction system. The result is a set of **Unstructured Tuples**, consisting of a set of unique keys plus affiliated values and possible attribute labels. We then use these tuples to choose an **Output Schema**, and thus create a domain-appropriate structured database. The **InfoExtraction** stage is provided by an external system, but the **SchemaExtraction** stage is novel.

or in a user's query. Of course, some text (*e.g.*, poetry) might not contain any reasonable structure, but this text is also unlikely to be the subject of a structured query.

This paper discusses an ongoing research program at the University of Washington to explore structured queries over the Web text corpus. Combining techniques from databases, information retrieval, and artificial intelligence, we are investigating three different models for processing structured text queries. They are quite different, but all involve some amount of information extraction and a novel query language. The models differ in how much structure they attempt to extract from the text, and how much they gather from the query.

Although we believe some of our query models present use scenarios that are quite compelling, it is still too early to determine a single best method for querying Web text. Indeed, it may be that different applications will require different query models.

2 The Schema Extraction Model

The **Schema Extraction Model** is the one described in our example in the introduction. The goal is to examine the corpus of Web text and create a "structured overlay" that describes data contained in the text. This structured overlay is essentially a relational database consisting of values found using an information extraction (IE) system. Users can then query this structured overlay using a fragment of SQL.

Figure 1 shows the execution pipeline. We start with a series of input texts; in the example here, some movie reviews. We then run an IE system, such as TextRunner, KnowItAll, or Snowball, over the text to obtain a series of fact-like assertions [2, 7, 1]. For a given identified object (*e.g.*, `Frenzy`) we have a series of extracted values, each of which has at least one accompanying attribute name (for `Frenzy`, perhaps values `1972` and `Alfred Hitchcock`, with attributes `year` and `director`). We call the extracted sets of values for one object an **unstructured tuple**.

These IE systems have shown to be quite effective at extracting small pieces of information embedded in text. For example, the KnowItAll system used the Web to compile a list of US states with precision 1.0 at recall 0.98 (that is, it made no incorrect guesses and missed only one of 50) and a list of countries with precision 0.79 at recall 0.87 [6].

We would like to assemble these unstructured tuples into a relational database that will form the structured overlay. However, the unstructured tuples do not have a uniform set of attributes, so the correct schema is unclear. Although most pages in a domain will contain similar information (here, `title`, `year`, etc), many will have missing values or off-topic extraneous ones. These missing and spurious values may be due to variance in the original text, or due to the inevitably-flawed IE mechanism. Some attributes will appear in nearly every unstructured tuple, whereas others will appear very rarely. The set of unstructured tuples may even describe a subset relationship (*e.g.*, horror movies may share only some attributes with action movies).

How can we generate a high-quality schema for these tuples? One approach is to create a relational table for each unique attribute signature in the unstructured tuples. If the unstructured tuples had no variance in their attributes (*i.e.*, they are "perfect"), this naïve algorithm would work well. However, in practice the result is a baroque and unusable schema, containing many similar but non-identical tables. In a small experiment, we mutated 20% of the cells in a set of perfect unstructured tuples. The result was a single-row table for every input tuple. Clearly, such a schema would be unsatisfactory to any query-writer.

For real-world data, especially data derived from many distinct texts without any explicit structure requirements, there will be no single indisputably "best" schema. Even if all the input data had been generated with a single schema in mind, some fields will be absent and some eccentric ones will appear. We must be willing to accept a "lossy" database that drops some extraneous extracted values found in the Unstructured Tuples. We must also accept NULLs where the appropriate value simply could not be recovered.

Of course, there is no genuine data loss here, since the original Web text still survives. But the lost data is not present in the structured overlay, so it will be unavailable to structured queries. An imperfect structured overlay is regrettable, but the only alternative is a "lossless" one that is so complicated as to be useless for querying. Choosing the lossy output that is least offensive for query-writers is a major challenge of this approach.

The Schema Extraction Model places the entire structural burden on the Web text itself. All of its technical challenges arise from the automated schema design. The structured query language itself is no more interesting than SQL.

Of course, if we can somehow find an extremely large corpus of schemas, then a slight variant of the Schema Extraction Model is possible. Instead of computing a schema directly from the extracted values, we can use these extractions to rank all schemas in the corpus, and then choose the best one. This schema corpus would have to be much larger than any corpus we have seen; it might be possible to compile it by crawling structured HTML tables, deep web services, etc.

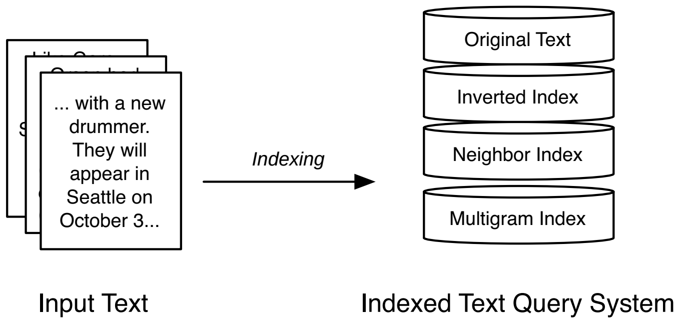
3 The Text Query Model

For the **Text Query Model**, consider a slightly different scenario. Imagine a student who wants to know when a favorite band will be playing nearby. The band's itinerary is available on its website. In a single query, the user would like to a) extract the city/date tuples from the band's website, b) indicate the city where she lives, c) compute the dates when the band's city and her own city are within 100 miles of each other. Of course, the user only wants to know about appearances in the future, even if old data is still on the website.

We might write that query as follows:

```
SELECT bandCity, bandDate
FROM ("http://thebandilike.com/**",
      ["to appear in <string> on <date>", bandCity, bandDate])
WHERE
bandDate > 2006 AND
geographicdist(bandCity, "Seattle") =< 100
```

In the Text Query Model, the user must indicate every relevant part of the query's structure. The `FROM` clause indicates a relevant set of web pages, an extraction expression that can be applied to the text to generate



... with a new drummer. They will appear in Seattle on October 3...
Indexing
Original Text
Inverted Index
Neighbor Index
Multigram Index
Input Text
Indexed Text Query System

Figure 2: In the Text Query Model, we do not do any information extraction ahead of time. Instead, we extensively index the Web text so that we can efficiently run users' extraction-driven queries.

a table with two columns, and labels for those columns. The `WHERE` clause tests the date to see if it is valid, declares the user's current city, and uses a built-in function to measure the distance between two cities. The user is completely explicit about which structural elements are useful for answering the query.

A naïve execution model requires downloading pages at all possible URLs that match the `FROM` clause, parsing them using the extraction phrase in the second part of the `FROM` clause, and finally testing the selection predicates on the resulting set of tuples. Previous Web query systems, such as Squeal, WebSQL, and W3QL, have not followed this algorithm exactly, but have fetched remote pages in response to a user's query (rather than, say, using preindexed and prefetched pages) [12, 10, 8]. The result is that this simple query would take an extremely long time to run over the entire Web, in stark contrast to search engine queries that routinely execute in under a second.

The standard indexing tools used by search engines and traditional relational databases cannot help us much. Consider a search engine's inverted index, which efficiently maps words to documents. We could start to execute the above query by using the inverted index to find documents that contain strings `to appear in` and `on` within the `thebandilike.com` domain. This inverted-lookup step, the basic component of all search engine queries, is very efficient. However, we still need to run selection predicates on the strings that match the `<string>` and `<date>` variables. These strings can only be found by examining and parsing the original document text. The inverted-lookup step gave us the document IDs, but fetching each such document from the disk will require a disk seek.

The resulting inverted-index query plan requires a very large number of disk seeks. Even if the selection predicates are so aggressive that they disqualify every possible result from being returned, the query needs to examine each document in the relevant domain. Query time will scale directly with the size of the document set being considered. The resulting query time should be much faster than the naïve approach, but still seems needlessly slow.

It is not as clear how to model this problem using a relational query engine, but it is still possible. One approach is to automatically create a relational table of extracted values using the `FROM` clause, and then execute the selection clauses against the resulting database. Even with a technique to efficiently find documents in the indicated domain, this technique is very expensive, requiring that we parse and process all the relevant text before we even begin running selection predicates. If our workload features repeated common `FROM` clauses, it might be profitable to index the extracted table, but such a workload seems unlikely.

We believe that a system that is built to process these mixed *extractive* and *structural* queries can do substantially better than any of the approaches listed above. For example, we could construct a *neighbor index*, which folds small amounts of the document text into an inverted index [3]. In the above example, such an index allows us to extract values for `<string>` and `<date>` immediately from the index itself, without loading the original

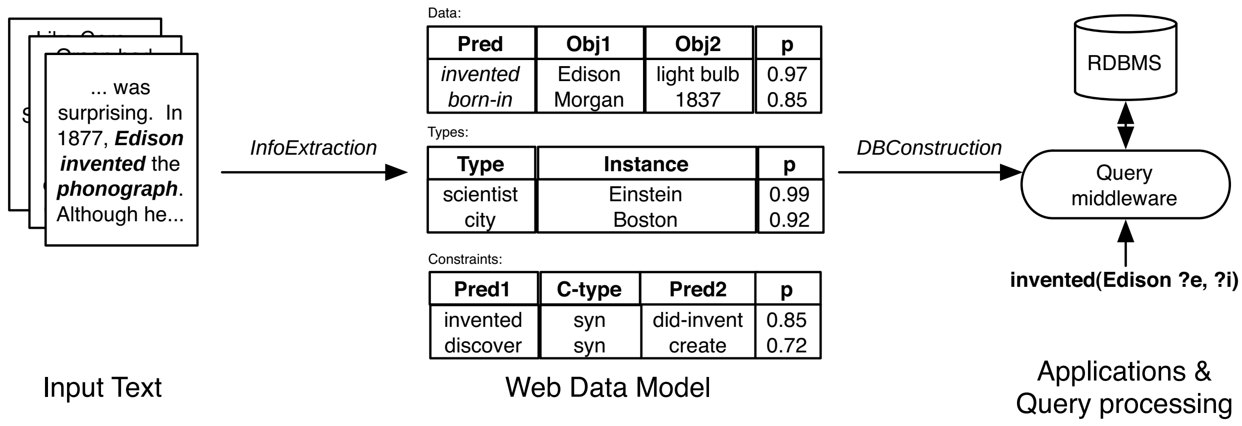

... was surprising. In 1877, Edison invented the phonograph. Although he...
Input Text
InfoExtraction
Data:
Pred | Obj1 | Obj2 | p
invented | Edison | light bulb | 0.97
born-in | Morgan | 1837 | 0.85
Types:
Type | Instance | p
scientist | Einstein | 0.99
city | Boston | 0.92
Constraints:
Pred1 | C-type | Pred2 | p
invented | syn | did-invent | 0.85
discover | syn | create | 0.72
Web Data Model
DBConstruction
RDBMS
Query middleware
invented(Edison ?e, ?i)
Applications & Query processing

Figure 3: In the **Extraction Database Model**, we use the output of several IE systems to create a series of tables for the **Web Data Model**. These tables describe very primitive data relationships, such as *facts*, *is-a* relationships, *synonym* constraints, and others. Each tuple retains a probability of being true. We place the tuples into a probabilistic database and run user queries over that database.

documents and incurring disk seek costs. We can push selection predicates down into the inverted index lookup, saving time when the predicates eliminate possible results.

The neighbor index is not appropriate for all queries, as it can only efficiently handle queries that return strings up to a fixed length. For example, we could pose a query whose execution requires a neighbor index much larger than the original document set, losing the efficiency of an inverted index. However, the neighbor index is an example of the kind of novel optimization technique we believe the Text Query Model both demands and makes possible. (Another possibility is the *multigram index*, which enables fast regular expression processing over a large corpus [4].) This is an active area for our research into structured query techniques for text.

4 The Extraction Database Model

The **Extraction Database Model** takes a middle approach. As with the **Schema Extraction Model**, we use information extraction techniques to discern useful tuples from Web text. However, we do not attempt to assemble these extractions into a full and coherent schema. Rather, we store the tuples until query time. The user's query contains additional structural information that describes the desired output schema. Unlike the previous two models, we obtain structural hints from both the text and the user's query.

Figure 3 shows the series of steps needed to create an Extraction Database query system. Note that our model of IE output is somewhat different. We assume that extractions come in one of a handful of forms. Most come in the form of fact triples, which consist of two *objects* and a linking *predicate*, and are similar to the unstructured tuples of the Schema Extraction Model. We also obtain types, which describe an *is-a* relationship between two objects. Also, we extract constraints, which describe various relationships between predicates. There are well-known mechanisms to generate each of these extraction types, such as TextRunner for trinary facts, KnowItAll for *is-a* relationships, and DIRT for synonyms [2, 7, 9]. Together, we call these extracted values the Web Data Model.

Finally, we also retain probabilities for all extractions. Because we can only determine a tuple's importance in light of a relevant query, we cannot threshold away low-likelihood extractions (as we could in the Schema Extraction case). We place all of these tuples into a probabilistic database, such as the MYSTIQ system [5, 11].

Users pose queries against the stored tuples using a SQL-like notation. It is easy to translate queries in the Extraction Database query language into a probabilistic SQL query over the Web Data Model tables. For example, in order to search for all scientists born before 1880:

```
SELECT s FROM scientist
WHERE
born-in(s, y) AND
y IN year AND
y < 1880 AND
```

This query returns just the value bound to variable `s`, which is constrained to be of type `scientist`, to appear as the first object in a fact with predicate `born-in`. The second object in that fact must be of type `year`. We also require that the value bound to variable `y` be less than `1880`. All Extraction Database queries elicit probabilistic tuples, which will be returned in descending order of probability.

Note that unlike the Schema Extraction Model, we do not try to discern before query-time whether, say, `born-in` is a salient attribute of an instance of `scientist`. The probabilistic store contains good and spurious extractions alike. We rely on the user's query to tell us that the `born-in` attribute is actually a good one to test. However, unlike the Text Query Model, the query does not do all the hard work; for example, the query-writer does not have to know about the actual text layout of real web pages. The Extraction Database Model tries to strike a balance between the two, by extracting facts that might be useful in the future, but not making firm decisions about the schema until the user's query provides more information.

Efficient probabilistic query processing is a difficult problem, described elsewhere at length [5, 11]. We exacerbate it here by the sheer scale of our extracted tables. A single high-quality page of text could generate several extractions for *each sentence*, depending on the actual IE systems and parameters being used. Query processing poses a substantial problem with the Schema Extraction Model, and is an area of current research.

5 Conclusions

Web text contains huge volumes of useful data, most of it now out of reach. Current keyword-driven search engines give access to single Web pages, ignoring the implicit structure in Web text that is apparent to every user. By recognizing this structure and making it available to query-writers, we can offer a query system that is hugely more powerful than current systems, all without asking publishers for any additional effort.

We have described three possible structured query models for Web text: Schema Extraction, Text Query, and Extraction Database. These systems offer different advantages regarding the information extraction technology required and the expressiveness of the queries. Each promises, in a different way, to take better advantage of the vast amounts of text available to us.

References

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Procs. of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[3] M. Cafarella and O. Etzioni. A Search Engine for Natural Language Applications. In *Procs. of the 14th International World Wide Web Conference (WWW 2005)*, Tokyo, Japan, 2005.

[4] J. Cho and S. Rajagopalan. A fast regular expression indexing engine. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[5] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Thirtieth International Conference on Very Large Data Bases(VLDB)*, 2004.

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-Scale Information Extraction in KnowItAll. In *WWW*, pages 100–110, New York City, New York, 2004.

[7] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[8] D. Konopnicki and O. Shmueli. W3QS - A System for WWW Querying. In *13th International Conference on Data Engineering (ICDE'97)*, 1997.

[9] D. Lin and P. Pantel. Discovery of inference rules from text. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 323–328, 2001.

[10] A. O. Mendelzon, G. A. Mihalia, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1996.

[11] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007. to appear.

[12] E. Spertus and L. A. Stein. Squeal: A Structured Query Language for the Web. In *WWW*, pages 95–103, 2000.

The 23rd IEEE International Conference on Data Engineering (ICDE 2007)

April 15-20, 2007, Istanbul, Turkey

Sponsored by The IEEE Computer Society and Microsoft Research

http://www.icde20007.org

CALL FOR PARTICIPATION

ICDE 2007 Highlights

We have an exciting program designed to appeal both to researchers and to data engineering professions. It includes:

- 3 keynote speeches;
- 5 Advanced technology seminars (no additional fee);
- Presentations of 111 research papers and 11 industrial papers out of 659 submissions;
- 55 poster papers and 28 demos;
- 15 workshops in conjunction with the main conference;
- Banquet

Keynote Speeches

- Yannis Ioannidis, *Emerging Open Agoras of Data and Information,* April 17
- Ricardo Baeze-Yates, *Challenges in Distributed Web Retrieval,* April 18
- Laura M. Haas, *Information for People,* April 19

Advanced Technology Seminars

- Ehud Gudes, *Graph Mining - Motivation, Applications and Algorithms*
- Hanan Samet, *Techniques for Similarity Searching in Multimedia Databases*
- Ashraf Aboulnaga, Christiana Amza, Ken Salem, *Virtualization and Databases: State of the Art and Research Challenges*
- Yannis Ioannidis, Georgia Koutrika, *Personalized Systems: Models and Methods from an IR and DB perspective*
- Limsoon Wong, *An Introduction to Knowledge Discovery Applications and Challenges in Life Sciences*

Planned Workshops

- First International Workshop on Data Engineering and Social Networks, *April 15*
  http://www.bayareabrains.com/~Ensundaresan/icde2007-snworkshop.htm
- Workshop on Data Processing in Ubiquitous Information Systems, *April 15*
  http://www.cs.technion.ac.il/~ranw/DPUbiq/
- The Second IEEE International Workshop on Multimedia Databases and Data Management, *April 15*
  http://www.cs.fiu.edu/mddm07/right.htm
- The Third IEEE International Workshop on Databases for Next-Generation Researchers, *April 15*
  http://zakuro.fuis.fukui-u.ac.jp/SWOD2007/
- First International Workshop on Scalable Steam Processing Systems, *April 15*
  http://research.ihost.com/ssps07/
- Workshop on Data Mining and Business Intelligence, *April 15*
  http://www.dmbi2007.itu.edu.tr/
- Third International Workshop on Privacy Data Management, *April 16*
  http://www.ceebi.curtin.edu.au/PDM2007/
- First International Workshop on Ranking in Databases, *April 16*
  http://www.cs.uwaterloo.ca/conferences/dbrank2007/
- Second International Workshop on Services Engineering, *April 16*
  http://www.hrl.uoit.ca/~seiw2007/
- Second International Workshop on Self-Managing Database Systems, *April 16*
  http://db.uwaterloo.ca/tcde-smdb/
- Workshop on Text Data Mining and Management*, April 16*
  http://www.personal.psu.edu/qzz1/TDMM/
- Third International Workshop on Web Personalisation, Recommender Systems and Intelligent User Interfaces, *April 16*
  http://www.doc.ic.ac.uk/~gu1/WPRSIUI/WPRSIUI07/WPRSIUI07.html
- International Workshop on Ambient Intelligence, Media, and Sensing, *April 20*
  http://aria.asu.edu/aims07/
- Workshop on Spatio-Temporal Data Mining, *April 20*
  http://stdm07.uhasselt.be/
- First International Workshop on Security Technologies for Next Generation Collaborative Business Applications, *April 20*
  http://www.cs.cmu.edu/~jinghai/SECOBAP/

Registration Fees

W: Workshops only registration
CW: Conference and Workshops registration
C: Conference only registration
(until Feb 16, 2007/after Feb 16, 2007)

	W ($)	**CW ($)**	**C ($)**
Member	300/350	650/800	500/610
Nonmember	350/425	850/1050	625/760
Member Student	200/250	350/450	225/275
Nonmember Student	240/300	450/550	290/360

Welcome to Istanbul

The former capital of three successive empires Roman, Byzantine and Ottoman, the city Istanbul is a fascinating mixture of the past and present, old and new, modern and traditional. The museums, churches, palaces, mosques, and bazaars, and the sights of natural beauty seem inexhaustible. As you recline on the shores of the Bosporus at the sunset contemplating the red evening light reflected in the windows and slender minarets on the opposite shore, you understand, suddenly and profoundly, why so many centuries ago settlers chose to build on this remarkable site.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398