Querying Web Data – The WebQA Approach

Sunny K.S. Lam* and M. Tamer Özsu
University of Waterloo
School of Computer Science
{s6lam,tozsu}@uwaterloo.ca

Abstract

The common paradigm of searching and retrieving information on the Web is based on keyword-based search using one or more search engines, and then browsing through the large number of returned URLs. This is significantly weaker than the declarative querying that is supported by DBMSs. The lack of a schema and the high volatility of Web make "database-like" querying of Web data difficult. In this paper we report on our work in building a system, called WebQA, that provides a declarative query-based approach to Web data retrieval that uses question-answering technology in extracting information from Web sites that are retrieved by search engines. The approach consists of first using meta-search techniques in an open environment to gather candidate responses from search engines and other on-line databases, and then using information extraction techniques to find the answer to the specific question from these candidates. A prototype system has been developed to test this approach. Testing includes evaluation of its performance as a question-answering system using a well-known evaluation system called TREC-9. Its accuracy using TREC-9 data for simple questions is high and its retrieval performance is good. The system employs an open system architecture allowing for on-going improvements in various aspects.

1. Introduction

Managing and, in particular, querying Web data has been one of the main preoccupations of the database community in recent years. The issue is brought to the forefront as a result of the difficulties of finding information on the Web due to its characteristics [4]: wide distribution of data, high percentage of volatile data, large volume, unstructuredness, redundancy, inconsistency of redundant copies, heterogeneity of data representation, and data dynamism.

*Current address: VoiceGenie, Toronto, Canada; sunnykslam@yahoo.com.

These fundamental characteristics of the Web contradict the basic assumptions of database management systems (DBMSs), posing significant challenges. From the database perspective, the above listed Web characteristics translate into three major challenges:

- There is no schema that describes the logical organization of the data. Without it, it is not clear what the basis of a query model is. As discussed later in this paper, many of the research projects assume an edge-labelled graph representation of Web data [17, 1, 6], but these models are hard to maintain given the scale and volatility of the Web data. Furthermore, many of these graph models are really not schemas, but representation of instance-level relationships. There have been attempts to extract a schema from this structure [10], but these approaches are not likely to scale.
- DBMSs search the entire database and return precise answers. Due to the scale of the Web, it is questionable whether one can hope to search the "entire Web". Even search engines only provide answers from using the data that they have crawled and indexed.
- It is not clear what the proper Web query language and query primitives are. Many of the earlier works [18, 2, 14, 12] have devised SQL-like languages, but these are based on graph models of the data. If, as argued above, these models cannot be generated for most of the Web data, then many of these proposals would not be feasible. Consequently, many of these works have focused on the integration of *some* of the Web data and running queries over them.

In this paper we describe our initial work on a Web querying system called WebQA. WebQA is an ongoing project focusing on querying Web data without depending on a schema. We have so far concentrated on providing an effective way to answer short factual queries. The research objectives are the following:

1. We wish to be able to query the "entire" Web and not only the subset that can be extracted and captured in a

database. We are, of course, constrained by the capabilities of search engines and metasearchers in accessing the Web data.

2. In doing this, we do not wish to depend on the existence of a schema that can be extracted from the Web. It is possible, of course, to exploit the existence of categorizations of Web data (e.g., Yahoo categories) or the existence of ontologies for parts of the Web, although our research has not yet considered these issues.

3. We wish to return actual answers, not URLs as search engines do. Although our initial work does not address these issues, our longer-term objective is to be able to do complex processing of Web data (e.g., joins and aggregate computations).

4. We wish the querying system to easily scale with new data sources.

5. In contrast to traditional database querying, we wish to be able to accept and deal with fuzziness in both user queries and in the results that are produced. In this context, information retrieval measures such as precision and recall are important, although these are not easy to define for Web data, as we will argue later.

At the current stage of the project we are not concerned with continuous queries, which are persistent queries that allow systems to return answers when the answers become available [7]. An example continuous query is *"Notify me whenever the Waterloo's temperature drops below zero"*. We also are not concerned with systems that use Web data mining to find answers to queries. Web mining systems analyze all the user queries in the past and determine what other topics might be of interest to a user depending on the user's query. They allow the return of non-relevant answers as long as they may be of interest to the users. For our research, we only study systems that answer the user queries directly. Finally, we focus only on factual queries. For example, *"What is the population of Canada?"*. We are not concerned with procedural queries. An example of such a query is *"How do I make pancakes?"*. Nor are we concerned with works such as [5] that only return Web statistic information.

WebQA involves a number of different techniques from various different areas. We use question-answering (QA) techniques for categorizing queries. We deploy metasearch techniques for selecting search engines or other data sources. We employ mediator/wrapper techniques to standardize and integrate results from different sources. We exploit information retrieval (IR) techniques for filtering out unnecessary result pages from the selected sources. Finally, we use information extraction techniques for extracting answers from the filtered pages.

The rest of the paper is organized as follows. In Section 2, we summarize the existing approaches to Web data querying. WebQA system architecture and components are discussed in Section 3. The details of the existing system implementation are discussed in Section 4. The evaluation of the system, in terms of both effectiveness and efficiency, is the focus of Section 5.

2. Related Work

There has been significant research related to searching for data over the Web. There are two major identifiable directions: information retrieval approaches (search engines and metasearchers), and database-oriented Web querying.

Search engines and metasearchers perform keyword searches. A search engine is a system that takes, as input, a list of keywords from a user and returns a list of related hyperlinks (with descriptions) to the user. A metasearcher, on the other hand, performs a keyword search over multiple heterogeneous search engines, collects the answers, and returns a unified result to the user. It usually has the ability to sort the result by different attributes such as host, keyword, date, and popularity. There are serious limitations of using keyword search. The keyword expression cannot fully represent the user's intent. There are multiple ways to enter a list of keywords for the same query and a list of keywords can be interpreted in multiple ways. There is no direct relationship between a user query and the keyword expression. Finally, keyword search is not sufficient enough to post complex queries (e.g., *"What is the distance between Toronto and Waterloo?"*). WebQA uses search engines as part of the set of retrieval engines, but goes further as we discuss in subsequent sections.

The precision of keyword searches can be improved by using category search (which is also known as Web directory, catalogs, yellow pages, and subject directories). There are a number of public Web directories available: dmoz, LookSmart, and Yahoo. A Web directory is a hierarchical taxonomy, which classifies human knowledge [4]. Although, the taxonomy is typically displayed as a tree, since some categories are cross referenced, it is actually a directed acyclic graph. Categorization and categories are also useful in the context of WebQA, as will be described in subsequent sections. They can also be useful in defining the ranges of variables in a Web query language.

Database-centric approaches to Web querying cover a wide spectrum. Approaches that involve Web data integration propose to integrate part of Web data sources into a database as either virtual or materialized views and query over these views. Some example systems include Information Manifold [16], Araneus [3], and the more recent WSQ/DSQ [11]. These approaches have the advantage of enabling well-known database querying techniques to be applied to the Web data. However, the search is not performed on the Web data itself, but on the few data sources from which data are extracted and integrated. Scalability of these systems and their ability to handle dynamic data are issues of concern. In contrast, WebQA leaves the Web data at its sources and use search engines, followed by information extraction techniques, to produce query results.

Another database-centric approach to Web querying is to model Web data as semistructured data that is represented as a fairly simple edge-directed graph that ignores complex structured nodes (i.e., records) and ordering. Querying is performed over this graph. The well-known languages over semistructured data are Lorel [1], UnQL [6], and StruQL [8]. These languages provide constructs that can accommodate the lack of a rigid schema for the data. The graph structure fits the natural link structure of Web pages, but the approach assumes that the Web data is transformed into the graph representation, a step that is ignored in most of the existing work. If large portions of the Web are considered, generation of the graph is not trivial and the graphs can become quite complex. The complexity is also an issue for users who wish to query the graph-modeled data. An approach to simplify the structure is to abstract these graphs into DataGuides [10]. DataGuides are, in fact, schemas over the Web data that can be used for querying.

A third approach has been to develop a class of languages that specifically target querying Web data by taking into account the Web documents' content and internal structure as well as external links (thus producing hypergraphs). These languages also assume an edge-labelled graph model of the Web data, but the graph structure is more complex, as indicated above. Florescu, Levy and Mendelzon [9] make a differentiation between first generation languages and second generation languages. The first generation languages (e.g., WebSQL [18], W3QL [12], and WebLog [14]) combine content-based querying (a la search engines) with structure-based querying that is common in database systems. These languages treat Web pages as atomic objects that contain (or do not contain) text patterns and they point to other objects. This information is modelled using a database approach. For example, WebSQL uses a relational approach to model the Web data by producing two (virtual) tables: Document and Anchor. The former has one tuple for each document in the Web and the latter has one tuple for each anchor in each document.

The second generation languages (e.g., WebOQL [2] and StruQL [8]) go beyond first generation languages in two ways: First, they provide access to the structure of the Web objects that they manipulate (they model the internal structure of Web documents as well as the external links connecting them). Second, they provide the ability to create new complex structures as part of a query.

Web query languages operate on a more powerful data structure than the semistructured languages since they consider a hypertree that is ordered and that explicitly represents internal and external links allowing languages to exploit different link types. The ability to construct new complex objects is also helpful in producing results that are graphs themselves that can be used subsequently. However, these languages still expect the user to know either the graph structure or, similar to integration-based approaches, they depend on being able to capture the Web data in relational tables.

The approach that comes closest to WebQA is Mulder [13] that is an online question-answering system like WebQA. It also aims to solve short factual questions and returns exact answers. There are many differences between WebQA and Mulder, however. The research focus in Mulder is on natural language processing and the "understanding" of user queries. WebQA's focus is to investigate database-centric querying over Web data. Mulder uses a single search engine for retrieving online documents (in a sense, it is adaptation of Google for question-answering systems). On the other hand, WebQA has the ability to query multiple sources (search engines and online databases) for passage retrieval. These sources can be search engines, weather sites, and other Web sites that store facts. Furthermore, WebQA is more fault-tolerant, because if one source fails, there are other sources to support the passage retrieval. Finally, WebQA uses wrapper/mediator architecture, giving it further flexibility and scalability. We will discuss these differences further in the Conclusions section.

3. WebQA Architecture

The long-term objective of WebQA is to provide full query capability over Web data. Therefore, the system will include the traditional query evaluation components (Figure 1). The fundamental difference, however, is the absence of a schema for querying. Therefore, the system uses question-answer (QA) techniques to access Web data through search engines, Web data sources (e.g., CIA World Factbook) and other Web sites that can provide answers to queries (e.g., weatherunderground.com). QA systems allow users to pose natural language questions and return the natural language answers. The systems retrieve relevant documents and extract answers from these documents. Traditional QA systems use off-line corpuses of documents on which queries are executed. WebQA, on the other hand, accesses open data sources as listed above.

In the proposed system (Figure 1), an incoming query is parsed and, depending upon its type, the best set of resources that can provide the answers to the query are identified and the query is decomposed into multiple subqueries that can be submitted to each of these sources. Answer Col-

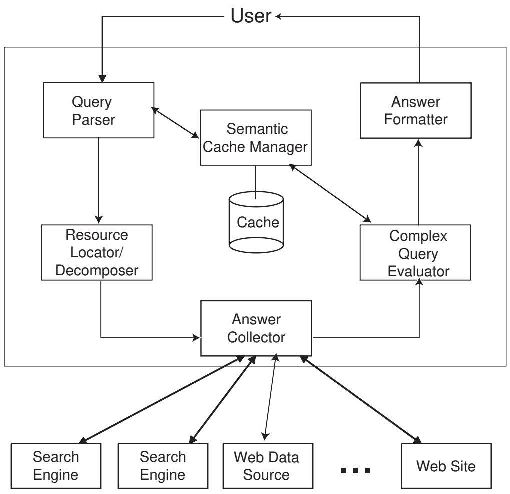

Figure 1. WebQA System Architecture

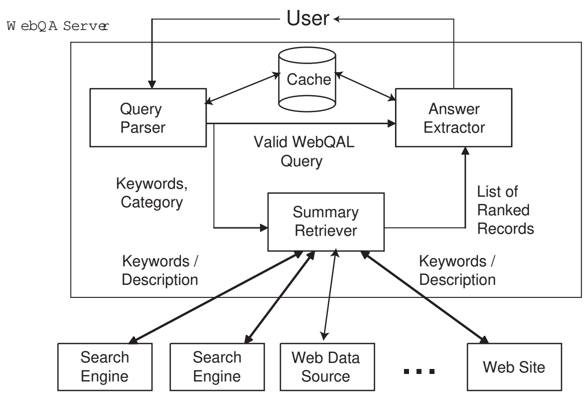

Figure 2. WebQA Prototype System Architecture

lector then interacts with these sources and obtains the results from each of these resources. The Complex Query Evaluator performs more complex operations (such as joins and aggregate computations) on the retrieved data.

We have initially implemented a prototype that focuses on the Answer Collector component depicted in Figure 1. This constitutes the basic infrastructure that needs to be put in place to access Web data. In the remainder, we focus on this prototype implementation and references to WebQA from now on will mean this prototype implementation in Java.

WebQA engine takes a short factual question and returns a list of short answers. It consists of three main components: the Query Parser (QP), the Summary Retriever (SR), and the Answer Extractor (AE) (Figure 2). This component-based architecture allows the system to be extensible and improvable as research continues.

WebQA currently provides a natural language interface to users, but converts this to an internal language called WebQAL that all system components understand. The intermediate language avoids the necessity of dealing with English ambiguity in other system components. The use of a natural language interface is due to the lack of a sufficiently flexible Web query language. Our research is now focusing on the appropriate Web query language constructs and semantics to allow for richer querying of Web data.

The general steps of executing a WebQA query are the following:

1. Given an input query, the QP component converts the natural language question to a valid WebQAL expression.

2. The WebQA engine checks whether or not the valid WebQAL expression is in its cache. If the query is found in the cache, then the answer is retrieved from the cache and returned to the user, otherwise, processing continues. (This is a very simple-minded implementation of a semantic cache.)

3. The WebQAL expression is passed to summary retriever (SR) that takes the expression and produces a list of ranked records (ranking is done globally). A record consists of a text passage from a source, the data source name where the passage comes from, and a local rank (rank within the source).

4. The answer extractor (AE) takes the list of records and extracts a list of answers produced by SR. AE's job is to take a list of ranked records and a WebQAL expression, and to extract a list of answers as output. Different extraction techniques can be applied.

5. The engine returns the list of answers to the specified user interface.

WebQA uses the client/server paradigm to interact with remote clients through a Web browser (Figure 3). Since a client accesses WebQA through a Web page, a Web server is necessary to handle the HTTP protocol. The Web server that WebQA uses is Jakarta Tomcat 3.3, which enables Java Server Pages (JSP). The Web user interface is written in JSP, which provides the ability to use Java to connect to the QA Server.

Once a Web server is started, it is able to accept any client requests (HTTP protocol). When a request comes, the Web server sends the request as a string to the QA server using Java's Socket. The job of the QA server is to accept any request from the Web server and return the answer back to the corresponding Web server. When a request comes from the Web server to the QA server, the QA server immediately generates a thread that handles this request. Again,

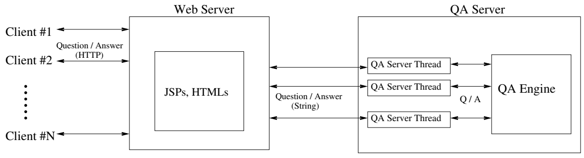

Figure 3. Client/Server Communication Architecture

all threads in the QA server are running concurrently. Each QA server thread asks WebQA's engine to get the answer of the given question. Finally the thread returns the answer back to the original Web server, which returns the answer back to the client as a new HTML page.

4. System Components

In this section we describe each of the components mentioned in Section 3. We focus on the function of each component and the general design considerations.

4.1. Query Parser

The Query Parser (QP) performs one of two functions depending on its input. If the user has specified a natural language (NL) query, then QP "*understands*" the query, categorizes it, and translates it into WebQAL. If the user query is specified in WebQAL, then QP functionality is restricted to the analysis and verification of the correctness of the WebQAL expression. The architecture of the QP is shown in Figure 4. If the given query is in natural language (NL), it passes the query to the Categorizer subcomponent. An example of a NL question is *"Who invented the telephone?"*. The Categorizer subcomponent determines the category, which, in this case, is *Name* (we discuss categories in more detail below). The category and the query are passed to the WebQAL Generator subcomponent. The WebQAL Generator generates a WebQAL expression (in this case name -keywords inventor telephone) and QP returns the WebQAL query. On the other hand, if the query is a WebQAL query, QP passes it to the WebQAL Checker subcomponent, which checks the validity of the given expression. If the query is valid, then QP returns the WebQAL, else an error condition is raised.

We should note that many QA systems use a full-fledged NL understanding system. The focus of these projects is *understanding* the user question and *extracting* the right answer, so the use of a heavy duty NL processor is warranted. In our case, the NL interface is not the research focus; in fact, it may (and will) be replaced by a more structured Web query language. Therefore, we have implemented a

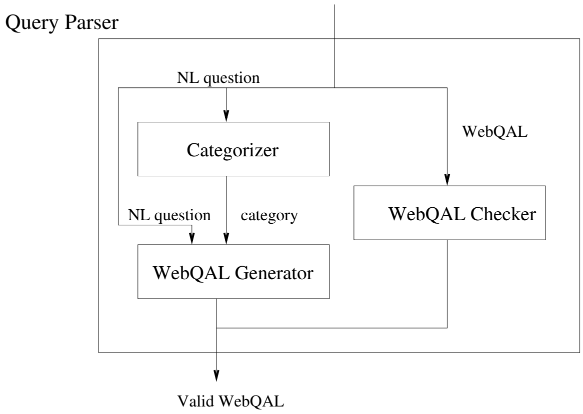

Figure 4. Query Parser Module

lightweight NL translator. This does not preclude the possibility of retrofitting a full NL understanding system as a front end to WebQA. As long as the front end performs categorization (most NL understanding systems can), it can be used with WebQA.

User queries are categorized into several classes to improve the accuracy of the system. The category information is used by all modules of the systems. Currently, the system operates with seven categories: *Name, Place, Time, Quantity, Abbreviation, Weather,* and *Other*, with the possibility of adding more categories in the future. Each category uniquely categorizes the output format. *Name* represents any names, such as the name of persons, companies, and universities, *Place* represents any cities, divisions (i.e., states, provinces, territories, or districts), countries, and continents, *Time* represents date, *Quantity* represents numbers with or without units, *Abbreviation* represents acronyms, *Weather* represents information of the weather, such as conditions, temperatures, barometers, wind speeds, dew points, humidities, visibility, sun rise time, sunset time, moon rise time, and moon set time, and *Other* category handles answers that cannot be categorized into any of the above categories. The reason for introducing categories is to improve system accuracy. More categories such as telephone number and stock quote can be added.

In our categorization, one question belongs to exactly one category. Categorization is rule-based where the rules are defined on the keywords in the questions. For example, a question that starts with "Who" or "Whom" is likely to ask for a name (e.g., *"Who invented the telephone?"*). However, rules are not this simple and can have many complications. For example the answer to the question *"Who was George Washington?"* should be "US president" or "first US president", not a name, even though the question starts with "Who". Thus, the rules need to be more sophisticated than simply looking at the question's keyword.

Since the main focus of this research is not on NL processing, we do not discuss the details of categorization further. We have tested the effectiveness of the categorization algorithm by considering TREC-9 and TREC-10 question sets. These are discussed in more detail in Section 5; for the time being it should suffice to point out that TREC-9 specifies a list of 693 questions and TREC-10 specifies a list of 500 questions that are used in question-answer system competitions. The effectiveness of WebQA categorizer was compared against manual categorization of these question sets, indicating an accuracy of 98.99% for TREC-9 and 92.2% for TREC-10.

The WebQAL Generator subcomponent takes a natural language query and the category, and generates a WebQAL expression. WebQAL, which stands for *WebQA Language*, is the common internal language used by WebQA components. The general syntax of WebQAL expressions is the following:

$\langle$*Category*$\rangle$ `[-output` $\langle$*Output Option*$\rangle$ `]` `-keywords` $\langle$*Keyword List*$\rangle$

The translation process involves stopword elimination (removal of terms such as *a, the*) and verb-to-noun conversion (e.g., *wrote* may be converted to *author*). The latter is important since search engines perform better using nouns as keywords. For example, given the question, *"Which country produced the most computers in the world?"*, the generator first eliminates the stopwords which results in *"produced most computers"*. Then the generator performs verb-to-noun conversion, which produces *"producer most computers"*. Finally, the generator plugs these into the keyword parameter list and get the following WebQAL expression:

```
place -keywords producer most computers
```

Some categories of questions can have answers of different types. Place is one of these classes where the output can be, for example, a city, a province, a state, or a country. For these classes, the categorizer uses a rule-based extraction of the output type. For the example query given above, the output would be a country, which would result in the following (final) WebQAL expression:

```
place -output country
-keywords most population world.
```

4.2. Summary Retriever

Summary Retriever (SR) takes a WebQAL expression and returns a list of ranked records as output (Figure 5). From the WebQAL expression the keywords that will be submitted to search engines or the particular search interfaces of other Web data sources are extracted.

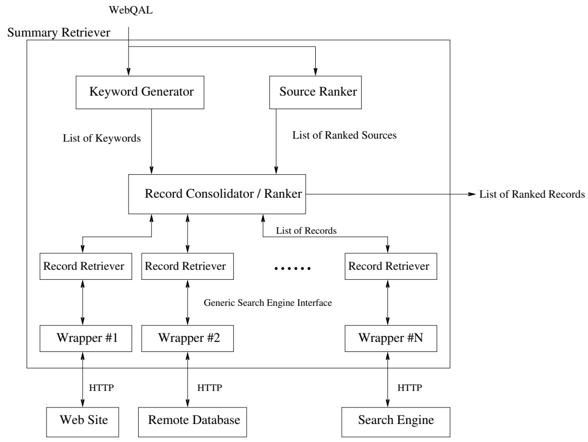

Figure 5. Summary Retriever Module

As our experiments (see Section 5) show, certain search engines and data sources are better at answering certain types of questions than others. The Source Ranker uses this knowledge and the category of the query to produce a ranked list of sources to be used in the search. At this stage, the source ranker is static with no learning component; it is our intention to improve the source ranker by allowing it to adapt itself both as it interacts with data sources and as it interacts with the users.

The list of keywords and the list of ranked sources are passed to the Record Consolidator/Ranker, which is the system component that interacts with the search engines and other Web data sources through the wrappers that are provided (the wrappers are discussed below). The Record Consolidator/Ranker submits the search to the sources included in the source list, consolidates the returned records, and ranks them according to its ranking algorithm. The output of Record Consolidator/Ranker, which is the output of SR, is a list of ranked records.

Record Consolidator instantiates a record retriever for each source that is identified as a useful source by the Source Ranker. Each record retriever requires two inputs. The first input is the keyword expression and the second input is the wrapper[1] for the record retriever's assigned source. A record retriever retrieves all the records from the wrapper and returns them to Record Consolidator. All record retrievers are implemented as threads, so they can run concurrently to save time. There are time limits for all the record retrievers, in order to produce answers in bounded time. When the system times out, Record Consolidator interrupts all active record retrievers and gets the records that have been retrieved so far.

[1] Since wrappers are well understood, we do not discuss them in this paper. However, we have wrapper templates for various data sources and search engines that makes it easier to accommodate newer sources when they become available.

Figure 6. The structure of a record

Attribute	Data Type
Source Name	String
Snippet	String
Local Rank	Integer

A *record* is a data structure with three components (Figure 6): a source name, a snippet, and a local rank. The source name is the name of the data source from where the information is obtained; the snippet is a piece of text that might contain the answer (different wrappers have different snippet structure, which we do not describe in this paper); and local rank is the rank of the record within the source. For example, in the case of search engines, the snippets are the short texts that are displayed together with a URL in the answer pages returned following searches. The rank is the ranking of the particular snippet within these answer pages. These records are used by the Answer Extractor (AE) component for extracting answers.

Record Ranker is dependent upon the extraction techniques which are described in the following section (Section 4.3). For the first prototype, we have implemented a simple algorithm that ranks the records based on the source ranking first and local ranking second.

4.3. Answer Extractor

Answer Extractor (AE) analyzes each record produced by the Record Consolidator/Ranker and extracts answers from these records. Information extraction is a rich field that has produced sophisticated techniques. The extraction algorithm that we use in this prototype is based on word frequency.

The records are analyzed and a candidate list of answers are identified. Each candidate is verified by checking it against the category and the output parameters that are captured in the WebQAL expression. The candidate identification is rule-based that uses a number of auxiliary databases that the system maintains (e.g., the list of countries). A score is associated with each rule that indicates the likelihood of the answer, and there is a pre-defined ordering of the rules based on the data source. A score is assigned to each candidate based on both the frequency of occurrence of the answer in the records and the score of the rule that adds it to the candidate list. The higher the score, the more likely is the candidate the answer to the user's query. The scores of the candidates are then re-arranged, if necessary. This may be important in cases where the candidate list includes both a long and a short version of the answer (e.g., "Bush" as well as "George Bush") in which case the shorter answer has a higher score because of its occurrence both on its own and inside the longer version. Finally, the system extracts the top x answers, where $x = 10$ in the current prototype.

5. System Evaluation

The current version of WebQA, as described in Section 4 has been implemented. We are naturally interested in its performance, both in terms of effectiveness and in terms of efficiency. The traditional effectiveness measures for this type of system are *precision* and *recall*. Precision refers to the proportion of "correct" results that the system returns, while recall refers to the proportion of the results that are returned. While these are suitable measures for systems that operate on closed sets of documents, they are hard to define for Web querying, since the denominators of their formulas cannot be easily determined. Therefore, WebQA is evaluated using the workload (queries) and measurement methodology of TREC-9[2], which is a well-known QA evaluation competition.

TREC-9 specifies a list of 693 questions that a system is expected to answer using a corpus of off-line documents. The answers are ranked responses, and, for each questions, a score of $1/n$ is assigned where n is the rank of the correct answer. Thus, the score is between 0 and 1. In the end, the average score of all questions is computed as the score of the system. Because TREC-9 evaluation system is designed for Information Retrieval systems operating over a closed corpus of documents, we have had to make certain adjustments to make it suitable for the open Web environment that WebQA targets. Space limitations do not allow us to discuss these adjustments that are discussed in [15].

We have set up a number of experiments for evaluating WebQA's accuracy and efficiency. We discuss a few of these experiments and their outcomes. All the experiments are run on a Pentium III 800 machine with 256 MB RAM running Windows 2000. They use an interpreter of Sun Microsystems' Java 2 SDK Standard Edition Version 1.3.1.

We measured the effectiveness of WebQA vis a vis the search engines using the TREC score. Each TREC-9 question is submitted to each search engine and the top five snippets returned by the search engine are extracted. The top five snippets are submitted to TREC-9 evaluation system. The system generates a TREC-9 score for the search engine.

The results of this experiment are depicted in Figure 7. These results are quite promising in that WebQA has higher TREC-9 score than the others. This means that the system provides some added precision to the results obtained by search engines.

The TREC score of WebQA compares favorably with those of other QA systems that were part of the TREC-9

[2]Ninth Text REtreival Conference.

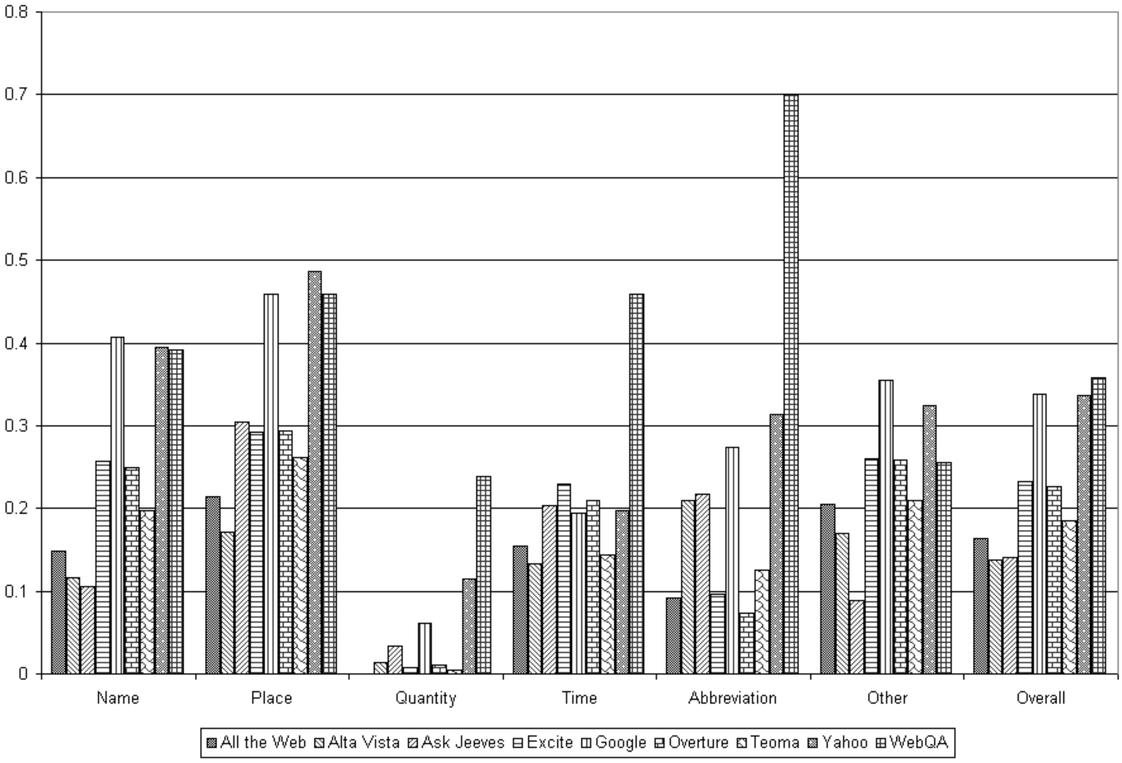

Figure 7. Effectiveness of WebQA

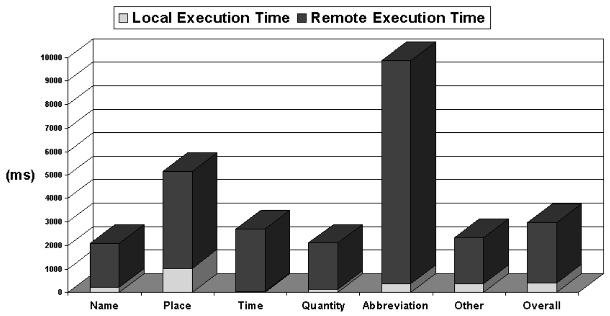

Figure 8. Response time for each category in WebQA

competition. WebQA's score ranks second among these systems. We should keep in mind that there are differences on how WebQA and traditional QA systems use TREC-9 for evaluation: they operate on a closed corpus of documents whereas WebQA operates in an open environment; the traditional QA systems retrieve 50 or 250 word text segments and the result is considered to be correct if the answer is found in this segment whereas we extract the exact answer.

A second experiment considered the efficiency of WebQA. For this experiment, we recorded the response time of each query including the interaction time with search engines and data sources. Therefore, the response time corresponds to client latency. Figure 8 shows the response time of WebQA for each category. All categories require less than 10 seconds for processing with a local execution time of less than one second. The bottleneck is the waiting time for the data sources. The *Abbreviation* category takes the longest, because it spends time waiting for the abbreviation page of World Factbook Web site. On average, WebQA takes around 3 seconds to process a query.

Since the prototype system implements a simple semantic cache, we conducted an experiment to see how the cache affects the response time. or each category, we run the experiment with various cache sizes and number of submitted queries. In order to test the cache, we uniformly pick a question from the TREC-9 question list.

Figure 9 shows the performance of cache hit rate by varying the cache size and number of submitted queries. Figure 10 shows the response time by varying the cache size and number of submitted queries. As shown in Figure 9, as the cache size increases, the cache hit rate increases as well. Once the cache size reaches 693 (i.e., the possible number of different queries), the cache hit rate stabilizes. Because

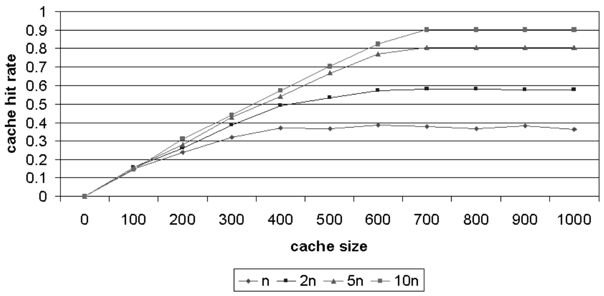

Figure 9. Cache hit rate v.s. cache size, where $n = 693$

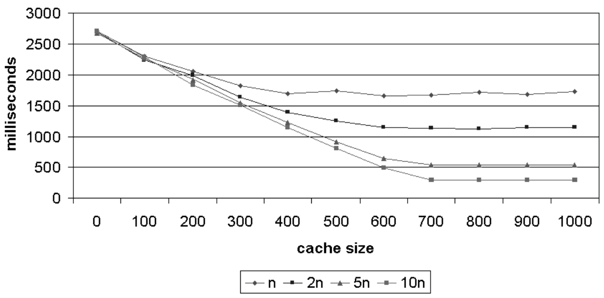

Figure 10. Response time v.s. cache size, where $n = 693$

the cache is filled up with all possible number of questions. The higher number of submitted queries, the higher cache hit rate. On the other hand, the higher number of submitted quires, the lower response time. As the cache size increase, the response time decreases.

6. Conclusions

In this paper we describe our current work on querying Web data within the context of the WebQA project. WebQA follows the question-answering approach and does not require a strict schema for querying. It is an open system with a wrapper/mediator architecture and has the ability to query different Web sources such as search engines, data sources (e.g. CIA's World Factbook) and Web sites (weather sites). It provides an uniform interface for users to query the Web effectively. The system is highly effective in obtaining results, gaining a TREC-9 score of 0.359, which would place it second among the traditional QA systems that have entered the TREC-9 competition.

WebQA is different from information retrieval-based Web search techniques in that it allows users to pose more sophisticated queries than simple keyword searches and it returns actual results rather than URLs to pages which *may* contain the result. It is also different from database-centric approaches in that it does not depend on integration or modelling of Web data. As indicated in Section 2, the only system that comes close to WebQA is Mulder [13]. Both systems accept short factual NL questions as inputs (this is true in the first prototype version of WebQA, but will change) and both return the exact answers as outputs. Both systems use search engines as their knowledge base. And both systems have similar main components which are common in traditional QA systems. However, there are major differences within each component of the systems. For query parsing, Mulder requires heavy NL parsing and WebQA takes a lightweight approach. Mulder classifies questions to only 3 categories while WebQA has 7 categories (with the ability to increase categories and be more specific). With more specific categories, we can increase accuracy by extracting answers with specific type. For example, WebQA has place category. If a user asks *"Which states has the most population?"*, WebQA verifies that the returned answer must be a valid state. Another issue we need to be concerned with is the accuracy of question classification. We have tried to compare our categorizer with Mulder's. The accuracy of our categorizer is higher than 90% and we have more categories. Unfortunately, [13] does not state the performance of the question classification. As for passage retrieval, Mulder only relies on a single search engine (Google), while WebQA accesses multiple search engines. In addition to increased effectiveness, this improves WebQA's availability and fault-tolerance. Furthermore, WebQA also queries relevant information sources that might contain answers to the question. Generally, these sources are more accurate than search engines.

There remains a lot to be done to achieve our goal of developing a full-fledged Web query system. As indicated earlier, we are investigating appropriate query paradigms and languages for Web data, looking into execution algorithms for more complex queries that involve joining results from multiple sources, or performing aggregates over retrieved data. The latter is particularly challenging since, as an open system, it is hard to know whether "all" the answers have been retrieved, and the denominators of some of the common aggregation functions (e.g., average).

Another line of investigation is the evaluation methodology for these systems. TREC-9 and other QA evaluation systems are designed for evaluating QA systems that search off-line data. One of major problems is that there is no evaluation system that specifically evaluate QA systems that search online data (i.e., the Web). The new evaluation system should be able to measure online QA systems' accuracy and efficiency. Since the data on the Web changes dynamically, the evaluation system should handle up-to-date answers.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journals of Digital Libraries*, 1(1):68–88, 1997.

[2] G. Arocena and A. Mendelzon. Weboql: Restructuring documents, databases and webs. In *Proceedings of 14th. International Conference on Data Engineering (ICDE)*, pages 24–33, 1998.

[3] P. Atzeni, G. Mecca, and P. Merialdo. To weave the Web. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 206–215, 1997.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, USA, 1999.

[5] Z. Bar-Yossef, A. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about Web pages via random walks. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB)*, pages 535–544, 2000.

[6] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[7] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proceedings of ACM SIGMOD International Conference on Management of data*, pages 379–390, 2000.

[8] M. Fernandez, D. Florescu, and A. Levy. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, 1997.

[9] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: a survey. *SIGMOD Record, 27(3):59-74*, 27(3):59–74, 1998.

[10] R. Goldman and J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 436–445, 1997.

[11] R. Goldman and J. Widom. WSQ/DSQ: A practical approach for combined querying of databases and the Web. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 285–296, 2000.

[12] D. Konopnicki and O. Shmueli. Www information gathering : The w3ql query language and the w3qs system. *ACM Transaction on Database Systems*, 23(4):369 – 410, December 1998.

[13] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the Web. In *Proceedings of 10th International World Wide Web Conference*, pages 150–161, 2001.

[14] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In *Proceedings of 6th International Workshop on Research Issues in Data Engineering (RIDE)*, pages 12–21, 1996.

[15] S. K. S. Lam. WebQA: A web querying system using the QA approach. Master's thesis, University of Waterloo, School of Computer Science, Waterloo, Canada, Spring 2002.

[16] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 251–262, 1996.

[17] J. McHugh, S. Abiteboul, R. Goldman, and J. Widom. Lore: a database management system for semistructured data. *ACM SIGMOD Record*, 26(8):54–66, 1997.

[18] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Journal on Digital Libraries*, 1(1):54–67, 1997.