The Web is the Database

Jayant R. Haritsa

Database Systems Lab, SERC
Indian Institute of Science
Bangalore 560012, INDIA
Email : haritsa@dsl.serc.iisc.ernet.in

Abstract. Search engines are currently the standard medium for locating and accessing information on the Web. However, they may not scale to match the anticipated explosion of Web content since they support only extremely coarse-grained queries and are based on centralized architectures. In this paper, we discuss how database technology can be successfully utilized to address the above problems. We also present the main features of a prototype Web database system called DIASPORA that we have developed and tested on our campus network. This system supports fine-grained querying and implements a distributed processing architecture.

1 Introduction

In 1998, there were about 320 million documents on the Web and this number grew to 800 million in 1999, comprising over 15 terabytes of information. During the same time, search engine coverage reduced from about 30% to about 15%. These statistics, reported in [13, 14], clearly indicate that the Web is experiencing tremendous growth – in fact, the anticipation is that it will grow by 1000 percent in the next couple of years [15] – and that search engines are proving unequal to the challenge, covering less and less of the document space. The database growth also means that search engines will return more and more documents to the user for the same query, resulting in the "data deluge" problem. This largely arises because search engines support only extremely coarse-grained queries and do not allow users to express their full knowledge about the domain space and thereby restrict the number of answers returned for the query. Finally, since search engines typically implement a centralized indexing and query-processing architecture, they are inherently not suited for scalability in terms of handling large amounts of data or high volume of user requests. All in all, it appears that search engines will soon run out of steam as the mechanism of choice for locating and accessing Web data.

In this paper, we investigate how database technology can be utilized to address the above data management and access problem. We consider various ways in which database technology and Web technology can be integrated and highlight the design challenges involved in a successful integration. We also summarize the main features of a system called DIASPORA, a new Web database system that we have developed which supports fine-grained querying and implements a distributed processing architecture. A Java-based prototype of DIASPORA is currently operational and is undergoing field trials on our campus network. Initial performance results indicate significant improvements in terms of both the quality of answers to user queries as well as the resources required to generate these answers.

The remainder of this paper is organized as follows: Background information on search engines and their limitations is given in Section 2. An overview of different approaches to combining Web and database technologies is made in Section 3. Our new DIASPORA system is described in Section 4. Finally, in Section 5, we present the conclusions of our study.

2 Search Engines and their Limitations

Search engines are currently the primary mechanism for accessing Web-based information, implementing index servers that provide URL references to documents. Users send query strings to these engines, which are applied against their indices to generate a ranked list of URLs of those documents that may have some related information. Well-known search engines include AltaVista, Excite, Yahoo, etc. While search-engines have contributed tremendously to the popularity of the Web as a publishing medium, they do suffer from a variety of limitations:

1. Each individual search engine covers only a small part of the Web, resulting in engine-specific answers to user queries. This forces users to query multiple search engines (each of which has its own data-entry format) in order to have reasonable coverage.

2. The query predicates are extremely coarse, operating primarily at the level of keywords. This makes it difficult for users to express sophisticated queries as well as to utilize any domain knowledge that they may have to eliminate irrelevant answers. For example, a query of the form *"Find the pages listing the faculty members from all the departments in Indian Institute of Science (IISc)"* that involves both *structural* predicates (restricting the search to web-sites in IISc) and *content* predicates (faculty member pages) is not expressible in search engine interfaces.
3. Search engines are based on a centralized architecture where all user requests are fed to a small set of servers. This means that as the Web grows and the number of users increase, these engines will become the bottleneck for efficient location and access of Web resources. In particular, the choice of a "data shipping" approach suffers from several disadvantages, similar to those already observed for traditional distributed database systems, including the transfer of large amounts of unnecessary data resulting in network congestion and poor bandwidth utilization, the client-site becoming a processing bottleneck, and extended user response times due to sequential processing.

In the remainder of this paper, we investigate how the above-mentioned limitations of search engines can be addressed by suitably integrating database technology into the Web query-processing framework.

3 Interfacing the Web and Databases

There are a variety of ways in which database technology and web technology have come together, which we have classified as "Database on Web" – Web is used as a communication medium; "Web on Database" – Web servers use database backends to host their content; and "Web is Database" – the Web forms the database, respectively. We discuss these frameworks in this section (for further material on database techniques for the Web refer [6]).

3.1 Database on Web

In this approach, the Web is used primarily as a *communication medium* for transporting queries and data between networked users and relational database servers. The main challenge here is the design of interfaces that facilitate embedding of SQL queries and their results into HTML and several such interfaces have been developed – for example, the WWW Connection interface for IBM's DB2 system [19]. This is explained in more detail below.

Organizations typically maintain their corporate information using database systems such as DB2 or Oracle. Using this backend database they provide different views of their organization to different groups of people (e.g. employees, customers and general public). These organizations exploit the low-cost and easily accessible feature of the Web. Web servers of such organizations are equipped with the *Common Gateway Interface* (CGI) techniques [23], enabling related users to access the corresponding view of the Database [19]. This is possible using HTML forms[20].

The mode of interaction is as follows: The Web server provides a starting HTML form to be filled and submitted by the user. This form is processed by executing the CGI-script at the server. The CGI-script produces another related HTML document, which may be a form again. This document corresponds to the view of the database/organization, valid as per the user identity/requirement. Communication in such a manner between the client's browser and the web server lets an organization provide access to its database to related users from anywhere on the web. This has also led to the concept of closed Internet, also known as *intranets*, within the organization.

The implementation of the Database on Web framework in *DB2-WWW Connection*[19] is shown in Fig. 1.

3.2 Web on Database

For larger organizations, having their web sites as a collection of HTML files is a major maintenance problem. They require to dynamically generate HTML documents, equip their sites with data/content query processing power, analyze their log records to improve and tune their performance, and organize their sites better.

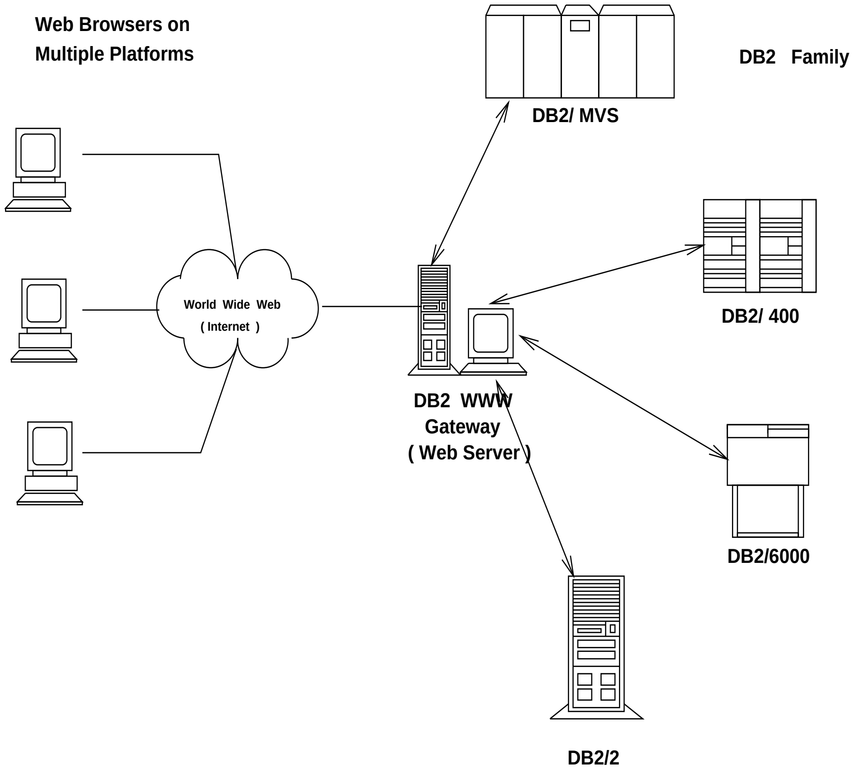


Fig. 1. Database on Web

This serves as motivation to move from a standard file system to a database system [8, 5]. The web server or HTTP daemon process is equipped with features similar to that of a database, to index HTML documents, in a manner that is transparent to the HTTP requests. These databases are required to handle complexities such as heterogeneous data (html files, images, applets, associated CGI-scripts, etc.), their indexing, querying, caching, etc.

Illustra's *Web Datablade System*, which provides a toolset to maintain database-driven web-sites [8], is shown in Fig. 2.

3.3 Web is Database

In this last approach, which we focus on for the rest of this paper, we view the Web *itself* as an enormous (and potentially the largest

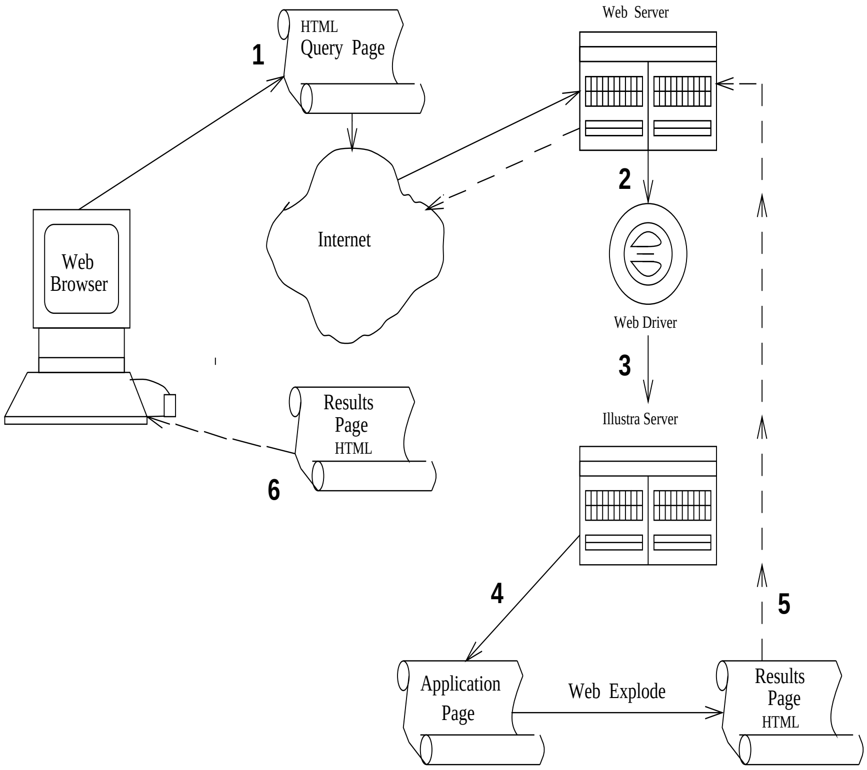


Fig. 2. Web on Database

in the world) database of information. A pictorial representation is shown in Fig. 3.

Porting classical database technology onto the Web is rendered difficult due to the heterogeneous, dynamic, hyper-linked and largely unstructured format of the Web and its contents. Further, the absence of a controlling entity equivalent to a database administrator makes it impossible to regulate the growth of the Web. In Section 4, we present a new Web database system that takes a first step towards providing an integrated and novel solution to these problems.

4 The DIASPORA System

In designing a database system that addresses the above challenges, the primary research issues that arise include the development of a

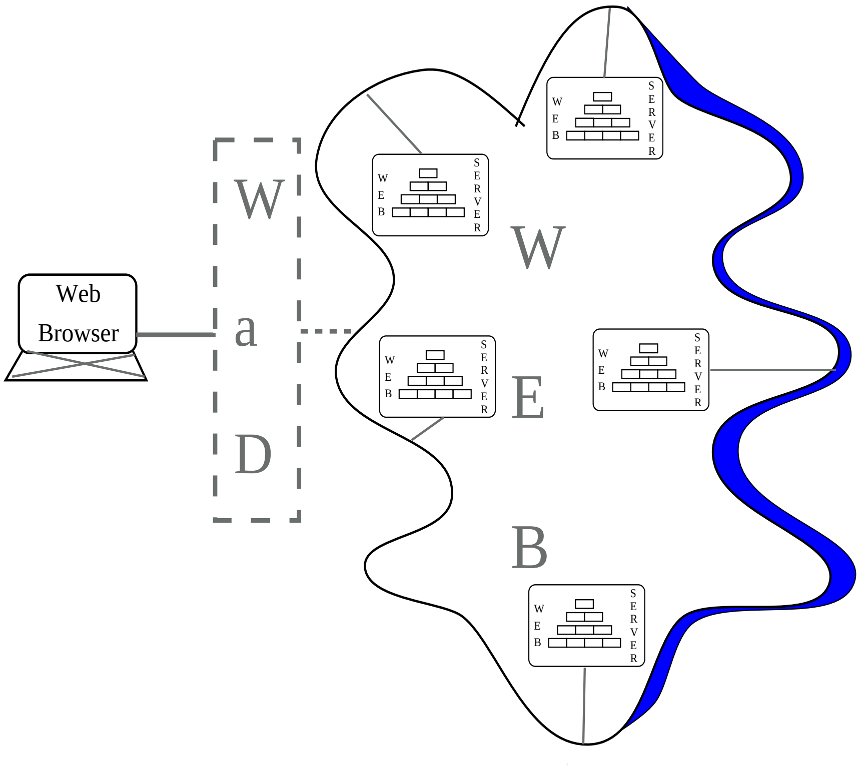


Fig. 3. Web is Database

data model that elegantly represents Web documents, a *query language* that enables users to easily process information represented according to this data model, and a *query processor* that can efficiently execute these user queries. We highlight here the main features of **DIASPORA** (DIstributed Answering System for Processing of Remote Agents), a new Web database system that attempts to provide an integrated and novel solution to the modeling, language and processing issues. The complete details of this system have been published in [21, 22]. A Java-based prototype of DIASPORA has been implemented and tested on our campus network. Initial performance results indicate significant improvements in terms of both the quality of answers to user queries as well as the resources required to generate these answers.

4.1 Data Model

DIASPORA implements a data model wherein each Web document is represented as a rooted, directed and edge-labeled graph [4], called *doc-graph*. At each site, the graphs of all the individual documents hosted at the site are integrated to form another rooted, directed and edge-labelled graph, called *site-graph*. A set of simple heuristics are used to "wrap" the data in the base HTML documents to conform to this data model.

Doc-graphs are intended to capture, in a hierarchical manner, the relationships between the elements in a Web document. While for XML, the meta-data is explicit in the tags, HTML documents pose more difficulties since they only have display information. We address this problem by using a set of heuristics to *infer* the meta-data – these heuristics utilize both the document structure and its contents. For example, section headings are regarded as metadata for the contents of the associated sections since they describe what the section contains. Similarly, the title of a document (enclosed in the <TITLE> tag) is used as the meta-data label for the entire document. The primary advantage of our approach is that it permits *automated* generation of the doc-graph, which is especially attractive when these graphs have to be generated for sites hosting a large number of documents.

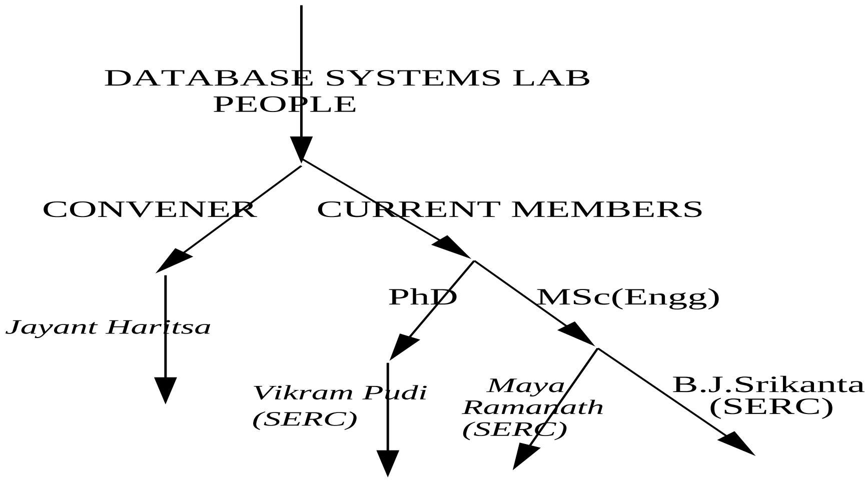
An example of the doc-graph generation process is presented in Figs. 4(a) and 4(b), which show part of an HTML document (from our lab's web-site) and its corresponding doc-graph (italics represent links), respectively.

We now explain how to build a *site-graph* from the set of doc-graphs associated with the documents hosted at a web-site. Like the doc-graphs, the site-graph is also a rooted, directed and edge-labelled graph, and is constructed using the following procedure:

- The site-graph is initialized to be the doc-graph of the home-page of the web-site.
- The "floating edge" corresponding to each "local link" (anchor that points to a document in the same web-site) in the home-page is terminated in the root of the doc-graph associated with the document pointed to by that link.

```
<HTML>
<HEAD>
<TITLE>DATABASE SYSTEMS LAB PEOPLE</TITLE>
</HEAD>
<BODY>
<H1>CONVENER</H1>
<UL>
<LI><A href="http://dsl.serc.iisc.ernet.in/~haritsa">Jayant Haritsa</A>
</UL>

<H1>CURRENT MEMBERS</H1>

<H2>PhD</H2>
<UL>
<LI><A href="http://dsl.serc.iisc.ernet.in/~vikram">Vikram Pudi (SERC)</A>
</UL>

<H2>MSc(Engg)</H2>
<UL>
<LI><A href="http://dsl.serc.iisc.ernet.in/~maya">Maya Ramanath (SERC)</A>
<LI>B. J. Srikanta(SERC)
</UL>

........
.......
```

(a) **Portion of an HTML Document**



(b) **Doc-Graph Representation**

Fig. 4. An HTML Document and its Graph Representation

– The above process is recursively executed for each of the documents that have been added to the site-graph, and terminates when all the documents reachable from the home-page have been included in the site-graph. Fig. 5 shows an example. Each box in the figure refers to a different document which has been converted into a doc-graph. The words in italics in the figure denote the labels of hyperlinks.

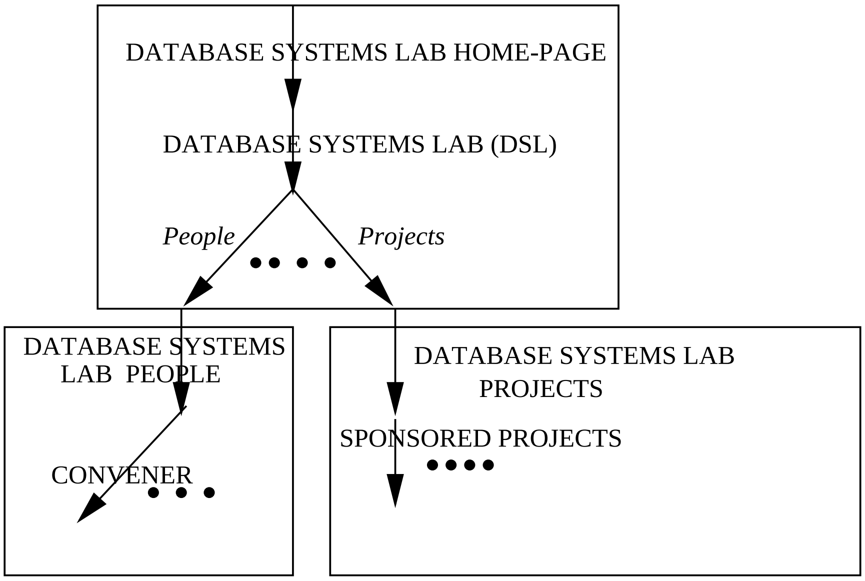


Fig. 5. Site-Graph Representation

It is perhaps natural to ask whether site-graphs of multiple sites should not be connected up together to form a "domain-graph". The reason we stop at building site-graphs is related to our query processing strategy, described later in Section 4.3 – since it adopts a query-shipping approach where queries visit the various web-sites, it is sufficient to maintain a site-graph at each site.

4.2 Query Language

We now move on to the query language supported by DIASPORA. The objectives in our language design were the following:

1. To enable the user to (a) express the *content* she is searching for through "hints" (in the form of keywords) to the query processor, and (b) express through "traversal expressions" any information she may have regarding the *structural* relationships among the web-sites where she wants the query to be processed.

   The ability to provide the query processor with hints and traversal expressions is critical to preventing the common problem faced by search engine users, namely, that of being deluged by a mass of results with no way of easily determining which few among these constitute the relevant set.

2. To present the results as a *weakly connected graph* that helps the user to "place" each result keyword – that is, to know where the keyword is located within the "big picture" of the Web document organization. This feature is especially helpful for users who are querying the Web database system in an interactive fashion, that is, using the results of a query as the basis on which to form more refined queries, and so on until eventually the desired information is reached. This is because the placement helps them determine the path, which if browsed, is most likely to lead to the desired information.

   For example, suppose the user has asked for publications on "databases" and gives the starting point for the search as the IISc homepage, the result graph would include a path from the IISc homepage to the SERC department homepage, from the SERC homepage to the Database Systems Lab homepage, from there to the publications page which lists the publications on "databases". Given such a placement, it will help the user determine whether the result is what she wants or not. Also, it will help her easily determine what *other information* she is likely to find if she decides to browse along that path.

To illustrate our solution to the above design criteria, the following is the expression in DIASPORA for the example query mentioned in the Section 2 (i.e. *Find the pages listing the faculty members from all the departments in IISc*):

```
1. SELECT
2.           { "*department*", "*faculty*" }
3. START
4.           http://www.iisc.ernet.in
5. WHERE
6.           DEFINE DeptLink AS LINK("*department*");
7.           DEFINE Dept AS KEYWORD("*department*");
8.           DOC_OF(START)   DeptLink   DOC_OF(Dept);
9.           DOC_OF(Dept)   G·G*1   SUBGRAPH_OF("*faculty*");
```

The purpose of this query is to "gather" information, and shows how the user's knowledge regarding the hyperstructure of the web can be used in formulating such queries. The user specifies her domain knowledge as follows:

– There is a path from the IISc homepage to the page listing all departments of IISc (departments page) through a hyperlink containing the word *department*. Thus, in order to locate the list of departments, start from the IISc homepage and traverse the hyperlink containing the keyword *department*.
– Each department listed in the departments page is a hyperlink which leads to the homepage of the department.
– Information about faculty members is found either at the department's homepage or a web-site directly reachable from the department's homepage.

The query contains a SELECT clause which states that the keywords of interest are "department" and "faculty". Then, each item of the user's knowledge is expressed as follows:

– Lines 6 and 7 of the query simply define a *hyperlink* (*DeptLink*) which contains the keyword "department" and a *keyword* (*Dept*) containing the term "department".
– Line 8 tells the query processor to start with the IISc homepage and then traverse the link *DeptLink* in order to find the document containing the keyword *Dept*.
– Line 9 tells the query processor to follow *at least* one global hyperlink from the current page and search for "faculty" in a resulting document reachable by following *at most* one global hyperlink from the resulting document.

In lines 8 and 9 we have used *DOC_OF* and *SUBGRAPH_OF*. These are collectively known as *Scopes of Traversal and Search*. When a scope occurs on the *LHS* of a traversal expression, it denotes the *traversal scope* and when it occurs in the *RHS* of a traversal expression, it denotes the *search scope*. Line 8 effectively states: "start from the *document* corresponding to START and traverse *DeptLink*, then restrict your search for *Dept* to the *document* reached". Line 9 states: "starting from the *document* corresponding to *Dept*, follow G·G∗1 and then search the *subgraph* of the destination reached for "*faculty*". In short, we make use of scopes in order to restrict or expand the search space and/or traversal space. A more detailed description of scopes is given in [21].

It is easy to see that the above query can be evaluated in a *centralized* manner at the user-site by importing the associated documents from each of the relevant web-sites, constructing a site graph and then processing the queries locally. This centralized mode of operation is a common feature of most previous Web database system proposals, including the SQL-like W3QL with interfaces to Unix tools [11], the declarative logic-based WebLog [12], the hyperlink-pattern-based WebSQL [17], as well as the OQL-based Lorel [1] and the graph-based UnQL [4] for semi-structured databases. However, as mentioned earlier, centralized approaches are inefficient from a variety of considerations including transfer of large amounts of unnecessary data resulting in network congestion and poor bandwidth utilization, the client-site becoming a processing bottleneck, and extended user response times. We therefore discuss next an alternative *distributed* approach.

4.3 Distributed Query Processing

In our distributed query-processing scheme, queries emanating from the user-site are *forwarded* from one site to another, the query is processed at each recipient site, and the associated results are returned to the user. Since our design ensures that the query forwarding does not require tight coordination from any "master site", it results in a highly distributed solution.

At an intuitive level, the distributed processing operates in the following fashion: The query is first sent to the sites corresponding to the *StartPoints* specified by the user in her query. Each of these sites completes its local processing of the query (which is some sub-query of the original query submitted by the user) and sends back the generated results, if any, to the user-site. Further, based on the structural hyperlink patterns (encoded as path regular expressions) in the query, it may *modify* the current query to reflect the completed processing of the sub-query and send the rest of the query to another set of sites. This set of sites is determined from the hyperlinks contained in the local site. These sites also perform similar query processing operations and the process continues until all the paths that match with the structural pattern have been fully explored and there are no more sub-queries remaining.

The above strategy is implemented through *QueryAgents*. A QueryAgent is a message that initially carries the entire query and its current processing state to the StartPoints. At each site the agent state is *updated* to reflect the movement and local processing of the query, and new QueryAgents may be generated to carry the unprocessed part of the query forward to other sites.

Results are *directly returned* from the query-site to the user-site. This is achieved by the user-site opening a *listening communication socket* to receive results – the associated port number is sent along with the QueryAgent. When a query-site wishes to communicate results, it utilizes the IP address of the user-site and the port number which came along with the agent to directly transmit the results to the user.

4.4 Determining Query Completion

Since, as described above, QueryAgents migrate from site to site without explicit user intervention, it is not easy to know when a query has *fully* completed its execution and all its results have been received – that is, how do we know for sure whether or not there still remain some agents that are active in the network. Note that solutions such as "timeouts" are difficult to implement in a coherent manner given the considerable heterogeneity in network and site characteristics. They are also unattractive in that a user may have to always wait until the timeout to be sure that the query has finished although it may have actually completed much earlier.

To address the above problem, we have incorporated in DIASPORA a special mechanism called the *CHT (Current Hosts Table)* protocol. The CHT protocol requires a minimal amount of synchronization between the query-sites and the user-site, but in return for this minor reduction in the decentralization of the processing, it ensures an effective and elegant means for determining query completion.

In this protocol, for each query submitted at a user-site, the local DIASPORA client process maintains a Current Hosts Table that keeps track of all the sites where the QueryAgents for this query are active. The attributes of the table are: (1) The URL of the EntryPoint (i.e. the starting HTML document) at the query-site, and (2) The state of the agent on arrival at the query-site. As described earlier in this section, after an agent arrives at a site and is processed, the local DIASPORA server determines the set of sites to which the new set of agents should be forwarded. Before forwarding the agents to these sites, the current site sends this "new-agent" information to the user-site in the form of a list of rows to be added to the CHT being maintained there. It also adds the URL of the EntryPoint and the (arrival) state of the agent that it received to the top of the list. When the user-site receives this list, it marks the entry in its CHT corresponding to the top-most entry in the list as deleted (signaling completion of query processing for the EntryPoint at the sending site) and inserts the list's remaining new-agent entries into the CHT. When all the entries in the CHT have been marked as deleted, it can be concluded that the query has been completely processed.

Note that only after the new-agent list is successfully sent are the agents forwarded to the next set of EntryPoints. The reason we process in this particular order is to ensure that the CHT at the user-site will *always have complete knowledge* about the sites at which the query is supposed to be currently executing and will therefore always be able to detect query completion. If the opposite order had been used, it is possible that the query may have been forwarded but the CHT not updated due to a transient communication failure between the current site and the user-site. This could lead to the possibility of the user-site wrongly determining that a query has completed when in fact it is still operational in the Web.

4.5 Query Termination

If a user decides to *cancel* an ongoing query, this message has to be communicated to all the sites that are currently processing the query. One option would be for the user-site to actively send termination messages to all the sites associated with the URLs listed in the Current Host Table. An alternative would be to purge the query locally at the user-site and to close the listening socket associated with the query – subsequently, when any of the sites involved in the processing of this query attempt to contact the user-site to return the local results, the connection will fail – this is the indication to the site to locally terminate the query. Note that since we insist that the CHT related information should first be sent to the user-site before forwarding the query to other sites, we do not run into the problem of termination messages having to "chase" query messages in the Web (this is similar to the problem of "anti-messages" chasing "event messages" in distributed optimistic simulation [7]).

4.6 Hosting Issues

An implicit assumption in the above framework is that a query processor capable of handling DIASPORA queries is executing as a daemon process at each site participating in the distributed execution of the query. At first sight, this requirement may appear unrealistic to fulfill – however, such distributed facilities are already becoming prevalent with the rapid spread of mobile agent technology [18]. Similar architectures have also been successfully implemented in the Condor distributed job execution facility [16] (now productized by IBM and called LoadLeveler). Further, even if some sites were to refuse to participate in this effort, we can always *revert* to the traditional centralized approach for the queries related to these sites. That is, we can have a *hybrid* query engine that is a combination of distributed and centralized processing.

Note also that for specific "domains" – for example, a campus or a company – that have a controlling authority, it may be quite feasible to have DIASPORA run at each site in the domain. Therefore, a starting point would be to use DIASPORA within such environments and then graduate to perhaps incorporating larger portions of the web.

Further, query-sites, especially those providing commercial or public services, may have a "selfish" motive for hosting DIASPORA – the fact that queries are run locally give it much more information about what users want and therefore can help it to structure its services much better. That is, the ability to do "query mining", to discover interesting patterns in what people are looking for can be the incentive for sites to participate in this cooperative endeavor.

4.7 Eliminating Query Recomputations

Due to the highly interconnected structure of the web, different agents of the original query may visit the *same* site in effectively the *same state* of computation following different paths. Note that if we do not detect these duplicate cases and blindly compute all queries that are received, not only is it a waste locally but subsequently the same sequence of steps followed by a previous agent will take place – in effect, we may have a "mirror" agent *chasing* a previously processed agent over the Web. This will also have repercussions at the user-site since the same set of results will be received multiple times and these will have to be filtered. In short, permitting duplicate query processing can have serious computation and communication performance implications.

From the above discussion, it is clear that each site should be able to evaluate the current state of an agent and also store this information locally in order to permit future comparisons. This is solved in Diaspora using an *Agent Log Table* that contains information with regard to agents that have previously visited the site, including the URL of the EntryPoint on which the agent is processed, the global identifier of the query, and the state of the agent.

When a new agent arrives at a query server, a new log table record is constructed for this agent and it is checked whether an equivalent entry already exists in the log table. If an equivalent entry exists, the agent is purged, otherwise, the new record is inserted in the log table, the agent is updated if required and then locally processed.

4.8 Reduction of Network Traffic

As discussed before, with DIASPORA no web resource is ever downloaded to perform a query operation over it. This is in marked contrast to the centralized approaches taken in search engines and in many of the previously proposed Web querying systems, including [17, 12, 11]. Apart from this, the additional optimizations are:

1. The agent results and the newly generated CHT information to be added to the CHT at the user site are *shipped together*. Further, if a query is received for multiple EntryPoints at a common website, all the associated results and corresponding CHT are *batched together* and sent to the user-site.
2. When forwarding agents, if the agents are to be sent to multiple EntryPoints that are all physically located at a common remote site, they are *bundled* together and sent *only once*.
3. Query termination is implemented passively, as described in Section 4.5, therefore not requiring additional termination messages from the query site to the sites currently hosting the agents of this query.

4.9 Performance Evaluation

Based on the above design, a prototype implementation of DIASPORA has been developed. The prototype is fully developed in Java – the details of the implementation are available in [21]. We evaluated our prototype of the DIASPORA system on a testbed of representative sites on our campus network. Our experimental results indicate that DIASPORA considerably reduces network traffic and improves user response times as compared to equivalent centralized systems.

5 Conclusions

In this paper, we motivated why search engines, due to their coarse query granularity and centralized architectures, may be expected to run out of steam in the future with the anticipated large increases in both the size of the Web and the query traffic. We discussed how database technology could be used to address these problems and mentioned various ways in which Web technology has been integrated with database technology, with special emphasis on the framework where the Web itself is treated as an enormous (and potentially the largest in the world) database of information. For this framework, we presented the highlights of the design of a new Web database system called DIASPORA that we have developed and successfully tested on our campus network. The system supports fine-grained content and structural queries and implements a distributed processing architecture. DIASPORA also opens up opportunities for mining user queries to improve commercial and public services offered by web-sites.

Acknowledgements: This work was supported in part by research grants from the Dept. of Biotechnology, Government of India. It is largely based on material drawn from the master's theses[9, 21] of my graduate students, Nalin Gupta and Maya Ramanath, whose results have been published in [10, 22].

References

1. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Weiner, "The Lorel query language for Semistructured Data", *Journal of Digital Libraries*, 1(1), 1997.
2. B. Adelberg, "NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents", *Proc. of ACM SIGMOD Conference on Management of Data*, 1998.
3. G. Arocena and A. Mendelzon, "WebOQL: Restructuring Documents, Databases and Webs", *Proc. of 14th ICDE Conference*, 1998.
4. P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu, "A Query Language and Optimization Techniques for Unstructured Data", *Proc. of ACM SIGMOD Conference on Management of Data*, 1996.
5. M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu, "Experiences with a web-site management system", *Proc. of ACM SIGMOD Conference on Management of Data*, 1998.
6. D. Florescu, A. Levy, and A. Mendelzon, "Database Techniques for the World Wide Web: A Survey", *SIGMOD Record*, 27(3), 1998.
7. R. Fujimoto, "Parallel Discrete-Event Simulation", *Communications of the ACM* 33(10), 1990.
8. J. Gaffney, "Illustra's Web DataBlade Module", *SIGMOD Record*, 25(1), 1996.
9. N. Gupta, "DISCOVER the Web-database", Master's thesis, Dept. of Computer Science and Automation, Indian Institute of Science, 1997.
10. N. Gupta, J. Haritsa and M. Ramanath, "Distributed Query Processing on the Web", *Proc. of 16th ICDE Conference*, 2000.
11. D. Konopnicki and O. Shmueli, "W3QS: A Query System for the World-Wide Web", *Proc. of 21st VLDB Conference*, 1995.
12. L. Lakshmanan, F. Sadri, and I. Subramanian, "A Declarative Language for Querying and Restructuring the Web", *Proc. of the 6th Intl. Workshop on Research Issues in Data Engineering*, 1996.
13. S. Lawrence and C. Lee Giles, "How big is the Web?", `http://www.neci.nj.nec.com/homepages/lawrence/websize.html`
14. S. Lawrence and C. Lee Giles, "Accessibility and Distribution of Information on the Web", `http://wwwmetrics.com`

15. J. Barrie and D. Presti, *Science*, 274, 371, 1996.
16. M. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations", *Proc. of 8th Intl. Conference of Distributed Computing Systems*, 1988.
17. A. Mendelzon, G. Mihaila, and T. Milo, "Querying the World Wide Web", *Journal of Digital Libraries*, 1(1), 1997.
18. D. Milijicic, W. LaForge, and D. Chauhan, "Mobile Objects and Agents (MOA)", *Proc. of the USENIX Conference on Object-oriented Technologies and Systems*, 1998.
19. T. Nguyen and V. Srinivasan, "Accessing Relational Databases from the World Wide Web", *Proc. of ACM SIGMOD Conf. on Management of Data*, 1996.
20. C. Pero, "HTML FORMS Tutorial", *Univ. of Illinois, Urbana-Champaign*, 1995, `http://robot0.ge.uiuc.edu/carlosp/cs317/cft.html.`
21. M. Ramanath, "DIASPORA: A Fully Distributed Web-Query Processing System", Master's thesis, Supercomputer Education and Research Centre, Indian Institute of Science, 2000.
22. M. Ramanath and J. Haritsa, "DIASPORA: A Highly Distributed Web-Query Processing System", *The WWW Journal*, 3(2), 2000.
23. "The Common Gateway Interface", Univ. of Illinois, Urbana-Champaign, 1995, `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.`