DIASPORA:A Highly Distributed Web-Query Processing System

Maya Ramanath Jayant Haritsa

Supercomputer Education and Research Centre

Indian Institute of Science, Bangalore 560012, India

{maya,haritsa}@dsl.serc.iisc.ernet.in

Abstract

Current proposals for Web querying systems have assumed a centralized processing architecture wherein data is shipped from the remote sites to the user's site. We present here the design and implementation of DIASPORA, a highly distributed query processing system for the Web. It is based on the premise that several web applications are more naturally processed in a distributed manner, opening up possibilities of significant reductions in network traffic and user response times.

DIASPORA is built over an expressive graph-based data model that utilizes simple heuristics and lends itself to automatic generation. The model captures both the content of Web documents and the hyperlink structural framework of a Web site. Distributed queries on the model are expressed through a declarative language that permits users to explicitly specify navigation.

DIASPORA implements a query-shipping model wherein queries are autonomously forwarded from one web-site to another, without requiring much coordination from the query originating site. Its design addresses a variety of interesting issues that arise in the distributed Web context including determining query completion, handling query rewriting, supporting query termination and preventing multiple computations of a query at a site due to the same query arriving through different paths in the hyperlink framework.

The DIASPORA system is currently operational and is undergoing testing on our campus network. In this paper we describe the design of the system and report initial performance results that indicate significant performance improvements over comparable centralized approaches.

1 INTRODUCTION

In the initial stages, the interaction between Web technology and database technology was limited to where the Web was used primarily as a *communication medium* for transporting queries and data between networked users and relational database servers. The main challenge here was the design of interfaces that would facilitate embedding of SQL queries and their results into HTML and several such interfaces were developed – for example, the WWW Connection interface for IBM's DB2 system [Nguyen and Srinivasan 1996].

Later on it was realized that the Web could *itself* be viewed as an enormous (and potentially the largest in the world) database of information. Porting classical database technology onto the Web is rendered difficult, however, 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 designing a database system that addresses the above challenges, the primary research issues that arise include the development of a *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 report, in this paper, on our design and implementation 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. 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.

1.1 Data Model

DIASPORA is based on an expressive graph-based data model that captures both the content of Web documents and the hyperlink structural framework of a web-site. This model is capable of handling both traditional Web formats such as HTML [Raggett 1997], which are focussed solely on document presentation, as well as currently developing formats such as XML [XML 1998], which also consider document semantics. For HTML documents, simple heuristics are used to *infer* semantic relationships among various elements in the document, thereby facilitating *fully automatic* generation of the graph (also see [Adelberg 1998; Ashish and Knoblock 1997]), whereas for XML documents the graph is built using the semantics explicitly encoded in the element tags.

1.2 Query Language

DIASPORA incorporates a declarative query language which lets the user specify "hints" to the query processor in the form of keywords. For example, a user who wants to find publications on warehousing published from the Indian Institute of Science (IISc) would use keywords such as "publication", "warehousing", and also specify that she wants the search to commence from the IISc homepage. The query processor then tries to *associate* these keywords and, while returning the results, give the user the context in which these keywords occur, thereby enabling her to judge which among the results contains what she is really looking for.

The query language also supports predicates on the *hyperlink structure* – these predicates can be utilized by users who have partial or full knowledge about how the information at web-sites of interest is organized and want to quickly extract information by "guiding" the query processor. For example, suppose the user wants to find the faculty members of all departments in IISc and knows that each department's homepage is reachable from the IISc homepage. She also knows that the faculty information will be provided either at the department's web-site itself or on a campus site that is reachable from the department's web-site. She can make use of this information by formulating a query whose processing starts from the IISc homepage and guide the query processor into following only particular hyperlink paths from this starting point.

1.3 Query Processing

The most novel feature of DIASPORA is its *highly distributed* query processing mechanism – prior literature has been mostly confined to centralized architectures. Our choice of a distributed approach is based on the premise that several web applications are more naturally processed in a distributed manner, opening up possibilities of significant reductions in network traffic and user response times.

Distributed execution is most attractive for supporting queries where there are predicates on both *structure* and *content* – for example, "find all the RealAudio song files that are reachable within two global hyperlink traversals from the IISc homepage". Here, the user knows the Web "neighborhood" in which to look for the information she needs but is not aware of the actual identities of all the sites in this neighborhood. In this situation, it would be extremely convenient and efficient if the query could be *automatically forwarded*, either directly or transitively, from the IISc server to the desired set of neighborhood sites, and the results directly returned from these remote sites to the site at which the user submitted the query.

We propose here a "query shipping" approach wherein queries emanating from a given site are forwarded from one site to another on the Web, 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.

A variety of interesting conceptual and practical issues arise when implementing a query-shipping approach including determining query completion, handling query rewriting, supporting query termination, transmitting results, and preventing multiple computations of a query at a site due to the query arriving at the site in different paths in the hyperlink framework. Our processing algorithm specifically addresses each of these issues.

1.4 XML and DIASPORA

In this paper, we focus primarily on HTML documents since they are the prevailing standard on the Web and will remain so for some time to come. However, the DIASPORA system can be easily extended to handle recent advances in markup technology such as XML and its host of derivative languages, which permit fine-grained querying on documents. In fact, we expect that when these formats become commonplace, distributed systems such as DIASPORA will become *even more relevant*. This is because many XML repositories are expected to be supported on relational engines to take advantage of the well-established power of these engines, with XML documents being materialized "on-the-fly" from these backends – two recent systems which support this "XML on RDBMS" architecture are described in [Deutsch *et al.* 1999] and [Shanmugasundaram *et al.* 1999]. For such systems, DIASPORA's server-side query processing strategy is especially attractive since it opens up the possibility of "pushing" queries on the XML documents into the relational engine, resulting in much faster processing.

1.5 Organization

The remainder of this paper is organized as follows: The data model and query language aspects of DIASPORA are described in detail in Sections 2 and 3, respectively. Its distributed query processing mechanisms are presented in Section 4 and a variety of associated performance optimizations are highlighted in Section 5. The results of an initial set of experiments with the system are presented in Section 6. Related work on the integration of Web and database technology is reviewed in Section 7. Finally, in Section 8, we present the conclusions of our study.

2 THE DIASPORA DATA MODEL

In this section, we describe the data model used in the DIASPORA system. Here, each Web document is represented as a rooted, directed and edge-labelled graph [Buneman *et al.* 1996], 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.

2.1 Doc-graphs

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.

The specific heuristics that we currently use for generating doc-graphs for HTML documents are the following:

- The title element forms the root of the doc-graph.
- A section heading is the parent for the contents of the section. If the contents of the section contain sub-sections, a tree structure results.
- Two elements are modeled as section-subsection based on their font size. For example, any element enclosed in a level i heading is a child of an element enclosed in a level $(i-1)$ heading. Hence, the text or subsection heading under a section heading is a child of that section heading.
- List items are enumerated and the set of items belonging to a common list are represented as siblings in the graph. With this approach, nested lists result in a tree structure.
- All anchor elements are edges with the same label as the anchor text and point to the edge corresponding to the destination of the anchor.

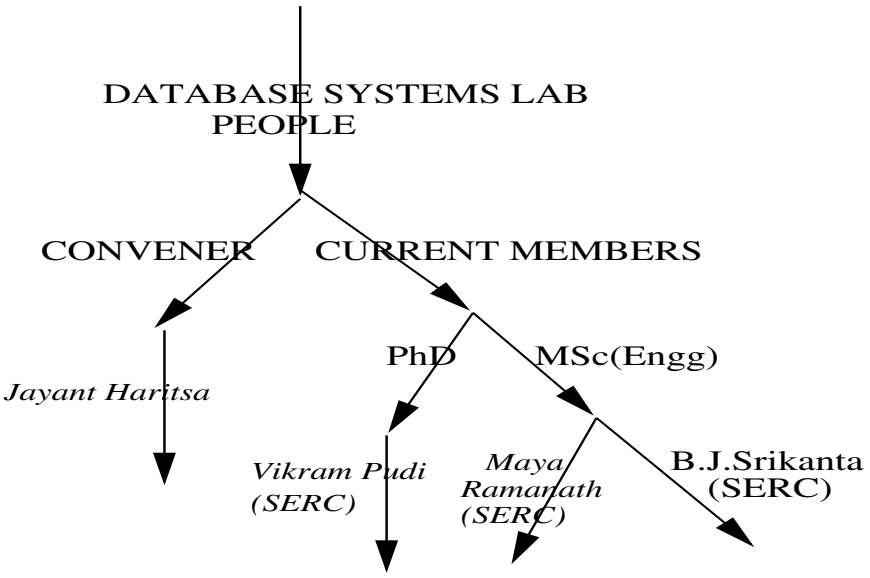
Note that additional heuristics, similar to those above, can be easily formulated to incorporate other HTML tags. Of course, these heuristics may not always result in the semantic interpretation desired by the document designer, but the few exceptions, if any, can be subsequently manually overwritten by the web-site manager.

An example of the doc-graph generation process is presented in Figures 1(a) and 1(b), which show part of an HTML document (from our lab's web-site) and its corresponding doc-graph (italics represent links), respectively.

Even though the inferred representation of a document may be accurate, it may not always be "complete" in the semantic sense. For example, it is the user's interpretation of the graph which can tell her that the current members of the lab include the convener even though "CONVENER" is not a child of "CURRENT MEMBER".

```
<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**

Figure 1: **An HTML Document and its Graph Representation**

In an XML equivalent, the information would perhaps have been organized more accurately with markup tags such as <MEMBER>, <FACULTY>, <STUDENT>, etc. which precisely describe the content and therefore make it suitable for automatic processing. But, in HTML, such precise descriptions are not to be expected since the emphasis is only on information display. However, it is still possible to answer queries like "who are the *people* in *Database Systems Lab*" and "who is the *convener* of the *Database Systems Lab*", etc.

2.2 From Doc-graphs to Site-graphs

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.[1)]
- 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. Figure 2 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.

At this stage, it is 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 – 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.

3 THE DIASPORA QUERY LANGUAGE

Having described DIASPORA's data model in the previous section, we now move on to describing its associated query language. The objectives in our design of the query language are 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.

[1)] Note that a local link can always point "back" in the graph leading to cycles. We do not eliminate such cycles.

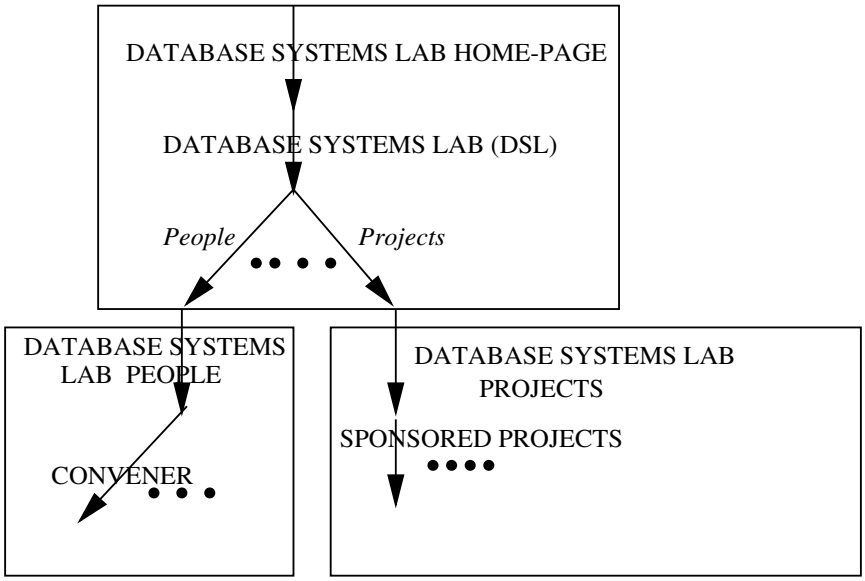


Figure 2: **Site-Graph Representation**

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.

3.1 Definitions

Due to space limitations and for readability, we have chosen to introduce the query language through a single query, rather than exhaustively defining all the language features, which are available in [Ramanath 2000]. Before we

describe this query, we first define the following terms, some of which were introduced in [Mendelzon *et al.* 1997]:

Hyperlinks: Hyperlinks (or simply links) are classified into the following categories:

1. Interior (I): A link whose destination is within the same document;
2. Local (L): A link whose destination is a different document but within the *same* web-site;
3. Global (G): A link whose destination is a document which resides on a different web-site;
4. Null (N): Denotes a null path, that is, the document itself.

An additional category of links are the user-defined links wherein existing document links are selected based on their label, source, destination, or category (I, L, G or N), or some combination of these attributes.

Path Regular Expression: A Path Regular Expression (PRE) is defined as follows:

- A link P belonging to one of the categories defined above is a valid PRE.
- Given a PRE P, then $P * [n]$ is also a PRE where the $*$ indicates repetition (if n is specified, the repetition is limited to a maximum of n – otherwise, we assume n to be some finite value).
- Given two PREs, P and Q, then $P|Q$ and $P \cdot Q$ are also PREs, where $|$ and $\cdot$ denote alternation and concatenation, respectively.

StartPoints: A StartPoint corresponds to an edge in a site-graph. This edge is determined by the URL specified by the user as the starting point of the query-processing.

3.2 Example Query

Our example query asks the question:

Find the pages listing the faculty members from all the departments in IISc

which translates to the following equivalent expression in our query language:

```
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 link from the current page and search for "faculty" in a resulting document reachable by following *at most* one global link 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 $G \cdot G * 1$ 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 Section 3.3.

3.3 Semantics of the Query Language

We first define the *EntryPoint* of a query that arrives at a site as the edge of the local site-graph from which the processing of the query starts at that site. While the StartPoint of a query is determined by the URLs provided by the user in the START clause of the query, the EntryPoint is determined dynamically, as the query is processed, modified and

forwarded from site to site. For example, suppose the processing at some site started from http://www.iisc.ernet.in, then the EntryPoint of the query would be the edge corresponding to the root of the site-graph for the IISc web-site. On the other hand, if the processing started from http://dsl.serc.iisc.ernet.in/people.html, then the EntryPoint would be the root of the *document* corresponding to the URL. This would be the edge in the site-graph for the DSL (Database Systems Lab) web-site corresponding to the root of the doc-graph of the document specified by the URL. Similarly, we would enter some arbitrary point in the site-graph of a site when we traverse a global link. Note that all StartPoints are also EntryPoints.

We now briefly describe the semantics associated with using the query language. In particular, we define *Search Scopes* and *Traversal Scopes*.

3.3.1 Search Scopes

When we say that the scope of search is the *DOC*, we mean that we restrict our search to the document at which the query has entered. If the query has entered at some intermediate point in the document, then we only search the subgraph from that point on, but do not move out of the document. Similarly, *SUBGRAPH* scope corresponds to the entire subgraph with the current entry point as the root[2)] and *SITE* scope would correspond to the entire site, regardless of where the query's EntryPoint was.

3.3.2 Traversal Scopes

We can now define the scope for traversal in the same way. When there is a link of type *T* to be traversed from *DOC* scope, we search only the current document for links of that type. If the scope is *SUBGRAPH*, we not only traverse the link of type *T* from the current document, but from the entire subgraph which can span multiple documents. Again, when the scope is *SITE*, we extract all links pertaining to the link type *T* from the whole site-graph and traverse them.

In the example query above, we made use of different scopes to search for the results. This works as follows: after the document containing *Dept* is found, line 9 tells us to start from *DOC_OF(*Dept*)* which is *DOC* scope and traverse one global link, again from *DOC* scope and search a *subgraph* to find "*faculty*". Thus, after traversing the global link, the scope for searching is not just the document pointed to by the link, but the entire subgraph from the point of entry. In effect we have a predicate which operates on a document and a subgraph. Similarly, predicates which operate only on documents, a document and a site, a subgraph and a site, etc. can be formulated.

The complete description of the query language, which has considerably more functionality than that described here, as well as a comprehensive set of example queries are available in [Ramanath 2000].

[2)]It is possible that the subgraph has "backpointers" which may point back, in the worst case, to the root of the site-graph. However, we have chosen not to eliminate such pointers since it might considerably change the results of the query depending on the EntryPoint.

3.4 Formal Query Specification

The use of traversal expressions impose an order in which the query is to be processed. For example, the example query in Section 3.2 can be broken up into two sub-queries, the first one which searches for "department" and the second which searches for "faculty". The first sub-query is processed after traversing a PRE from the StartPoint and the second sub-query is processed after traversing another PRE, the links for which are extracted based on the results of the first sub-query. Thus, we can formally denote a query as an alternating sequence of sub-queries and path regular expressions. That is,

$$\mathcal{Q} = \mathsf{S}\ p_1\ \ q_1\ \ p_2\ \ q_2 \cdots p_n\ \ q_n$$

where S is the set of StartPoints from where the query begins its execution, q_k is the k^{th} sub-query, and p_k is the PRE to be satisfied after q_{k-1} is evaluated and before q_k can be evaluated.

3.5 Query Result at a Site

We describe here how the result for a sub-query is generated from the local site-graph. Assume that the sub-query q_i being processed at this site contains keywords $K_1, K_2, \cdots, K_n$ in its SELECT clause. In addition to this, let the conditions (such as those involving functions on documents) to be satisfied be $c_1, c_2, \cdots, c_m$. The query processing now includes the following:

1. Let $\mathcal{G}$ be the site-graph.
2. Identify the EntryPoint of the current sub-query q_i and let this edge be $\mathbf{e}$.[3)]
3. Search the set of edges in the search scope of q_i for the result edges. Let the set of edges which should be included in the result be E_{result}. This set of edges will include only those edges whose label is the superstring of at least one K_i *and* which are contained in the search scope of q_i *and* which satisfy the relevant conditions c_i.
4. Next, in the traversal scope for q_{i+1}, find the hyperlinks to be traversed in order to process the next sub-query. That is, find the set of edges which correspond to the traversal as specified by p_{i+1}. Let this set be $E_{forward}$.
5. We now have the set of edges which *must* be included in the result graph. Let this set be $E_{all} = E_{result} \cup E_{forward} \cup \{\mathbf{e}\}$.
6. In order for the result to be seen *in context* by the user, we need to form a *weakly connected graph* in which the context of the edges in the result set E_{all} is shown. Note that the set of edges in E_{all} need not form a connected graph on their own.

[3)] As mentioned in Section 3.3, the EntryPoint is the edge from which the query processing at the current site starts.

Let E_{add} be the set of edges (with minimum cardinality) from the site-graph $\mathcal{G}$ such that the set of edges $E_{add} \cup E_{all}$ forms a weakly connected graph. This weakly connected graph is now the result graph $\mathcal{R}$.

For example, suppose we had a query which finds the publications on concurrency at the DSL web-site and the SELECT clause for this query contained the words "concurrency" and "publication". Running the above algorithm on the DSL web-site for this query will return a result graph, part of which is shown in Figure 3 (the italics in the Figure indicate hyperlinks). As this Figure shows, the *placement* of the keywords "publication" and "concurrency" in the DSL site-graph, helps the user determine whether this is indeed the answer she is looking for.

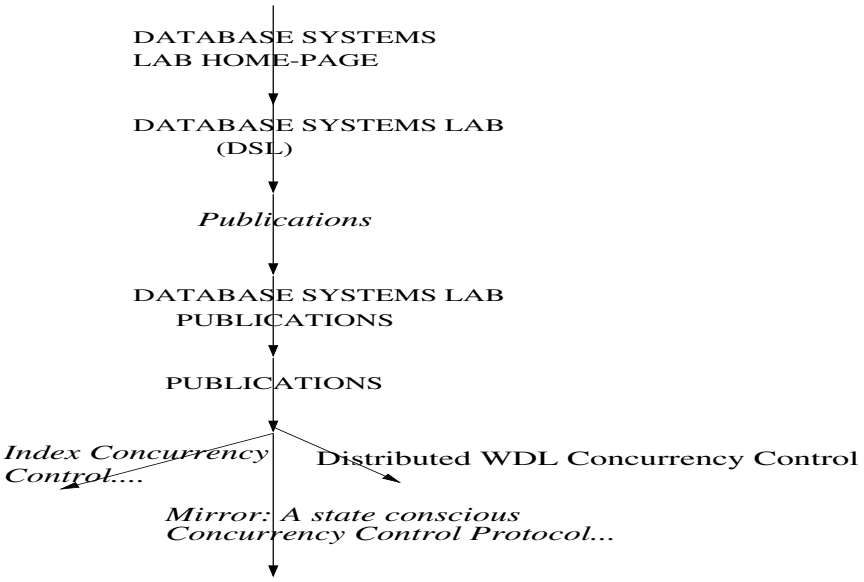


Figure 3: **Example Result for Query at the DSL web-site**

Given that we can produce results from individual site-graphs as described above, it is easy to see that the entire 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 is, for example, the mode of operation typically assumed in previous Web database system proposals.

However, as mentioned in the Introduction, this centralized approach is 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 an alternative *distributed* approach in the following section. This idea is also supported in the concluding remarks of [Mendelzon *et al.* 1997] – "It would also be interesting to investigate a distributed architecture in which subqueries are sent to remote servers to be executed there, avoiding unnecessary data movement."

4 DISTRIBUTED QUERY PROCESSING

In the previous sections, we described the data model and query language of the DIASPORA system. We now consider the issue of efficiently processing queries submitted to the system. In particular, we present our scheme for processing these queries in a distributed manner, wherein 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.

Our scheme is based on the formal query specification described in Section 3.4. 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 patterns encoded as PREs 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 PRE have been fully explored and there are no more sub-queries remaining.

4.1 Preliminaries

To help describe the query processing scheme, we need the following definitions:

User-site: The web-site at which the user submits the query.

QueryAgent: 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. For simplicity, we will use the word *agent* to refer to QueryAgents in the remainder of the paper.

Query-site: The web-site at which a query gets processed. For simplicity, we will often use just the word *site* when the context is clear.

4.2 Query Processing and Forwarding Scheme

A brief description of the functions the user-site and the query-sites perform are given next. The user-site simply sends the QueryAgent to each of the StartPoints. These are the first set of EntryPoints. The sites containing

the EntryPoints potentially have two roles to play:

ServerRouter: A site operates as a *ServerRouter* if it evaluates the current sub-query in the agent and forwards updated agents to the next set of sites based on the *next* PRE. This means that the next set of sites will potentially process the *next* sub-query.

PureRouter: A site operates as a *PureRouter* if it merely traverses the next link in the current PRE.

To make the above distinctions clear, consider the following examples of queries received at a site S_i:

1. N q_i L q_{i+1}: Here, p_i = N and p_{i+1} = L, and S_i acts as a ServerRouter since p_i is the null link, that is, S_i *evaluates* q_i and traverses link L of the *next* PRE, p_{i+1}.

2. G q_i L q_{i+1}: Here, S_i does not evaluate q_i since p_i = G does not contain the null link. Hence, S_i acts as a PureRouter and only traverses the link G.

3. G$*$**1** q_i L q_{i+1}. In this case, S_i not only evaluates q_i (since p_i = G$*1$ contains the null link) and traverses the link L which is part of the *next* PRE, p_{i+1}, but also traverses G, which is part of the current PRE. Thus, in this case S_i acts as both a ServerRouter as well as a PureRouter.

Depending on whether a query-site is a ServerRouter or a PureRouter, the steps taken by the query-site are as follows:

- if the site is a ServerRouter, then
  1. evaluate the sub-query[4)]
  2. return results to the user-site
  3. create a set of "clone" QueryAgents from the currently received QueryAgent for each of the sites containing the next set of EntryPoints to which it has to be forwarded to as determined by the PRE.
  4. for each next EntryPoint,
     - modify the PRE information carried by the clone to reflect the traversal of the query to the EntryPoint
     - include the URL of the EntryPoint as the destination in the clone
     - dispatch the clone to the site of the EntryPoint[5)]
- if the site is a PureRouter then process from step 3 above

For more detailed algorithms describing the functions of the query-sites and the user-site, refer to [Ramanath 2000].

[4)]Here, evaluating the sub-query amounts to evaluating the SELECT clause. Note that the parameters to the *search scope* of the current sub-query determine which keywords in the SELECT clause are relevant at this site.

[5)]Note that a QueryAgent needs to be explicitly "forwarded" only if the EntryPoint to be considered resides on a different site.

4.3 Returning Results to the User

In our scheme, 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, described below. 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.

4.4.1 The CHT Protocol

To describe the CHT mechanism, we first need to define the processing state of an agent. For our purposes, the state of an agent A, denoted by $S(A)$, is completely captured by the following:

num_q : The remaining *number* of sub-queries yet to be processed. Note that only the number is required, not the details of the queries.

$rem(p_i)$: The remaining part of the current PRE to be traversed before the next sub-query can be evaluated.

So, for example, $S(A)$ = (2, G · L), denotes that there are two more sub-queries yet to be processed and that the traversal path to the next sub-query is a global link followed by a local link.

For each query submitted at a user-site, the local DIASPORA client process maintains a table called the *Current Hosts Table* (CHT). This table 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 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 (no duplicates are allowed). 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 Entry-Points. 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.

The algorithms employed at the user-site and at the query-site for supporting query completion detection are available in [Ramanath 2000].

4.5 Construction of Results

The construction of results takes place in two phases. In the first phase, a sub-query q_i is evaluated at a query-site and a *result graph* is constructed. Thus, for each q_i, several result graphs from several different query-sites are returned to the user-site. In the second phase, which is executed after the entire query has completed (determined as described earlier in Section 4.4), the entire set of sub-query result graphs is connected together to form the set of final result graphs.

The general outline of the algorithm to construct the result graph is as follows:

- Each site constructs a result graph as outlined in Section 3.5.
- From each site which constructs such a result graph and forwards the agent(s), the following information is sent back to the user-site: (1) EntryPoint of the query at this query-site; (2) $S(A)$, the state of the agent when it arrived at this query-site; and (3) For each link traversed, ($S(A)_{new}$, destination of the link). This information is used in constructing the final result graph at the user-site.
- Once the user-site has the individual result graphs, and the auxiliary information from the query-sites, it con-

structs the final set of result graphs which relate the results from each of the sites.

4.6 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 [Fujimoto 1990]).

4.7 Related Issues

Having discussed the mechanics of the query shipping approach, we now comment on some related 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 [Milijicic *et al.* 1998]. Similar architectures have also been successfully implemented in the Condor distributed job execution facility [Litzkow *et al.* 1988] (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. In fact, as described later in Section 6, we have DIASPORA currently operational on our campus network.

Note also that 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 endeavour.

5 PERFORMANCE OPTIMIZATIONS

Having discussed the basic distributed query processing framework in the previous section, we now move on to highlighting the optimizations included in the DIASPORA system to minimize the computation and communication overheads involved in supporting this framework.

5.1 Eliminating Query Recomputations

Due to the highly interconnected structure of the web, different agents of the original query may visit the *same* site at the same EntryPoint following different paths. In this situation, there are two possible cases:

1. The agent arrives in a *different state* of computation as compared to previous agents of the associated query.
2. The agent reaches in effectively the *same state* of computation as a previous agent.

The above possibilities are illustrated in Figure 4, which is based on the following query: S (G * 2 | L) q_1 . Here, we see that site S_1 is visited by three different agents of the query, first at the EntryPoint *a* (box labelled X), then at the EntryPoint *b* (box labelled Y), and then again at the EntryPoint *b* (box labelled Z).

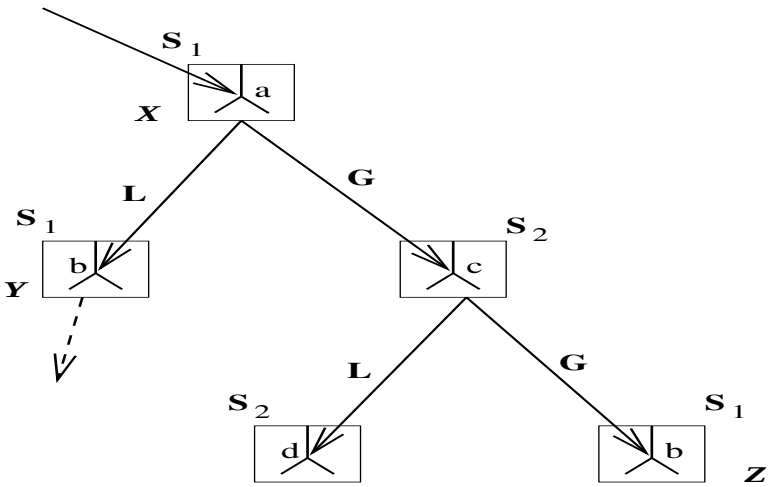


Figure 4: **Redundant Multiple Visits to a Site**

While evaluating the sub-query is mandatory in Y, it is obviously a waste in Z since the same query has been previously computed in Y at the same entry-point. Note that if we do not detect the 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. Our solution to this issue is described in the remainder of this subsection.

5.1.1 Agent Log Table

At each site, DIASPORA maintains a *log table* that contains the following information with regard to agents that have previously visited the site. Each log-entry is a tuple $[URL, QID, S(A)]$, with the following semantics:

URL : The URL of the EntryPoint on which the agent is processed

QID : The global identifier of the query

$S(A)$: The state of the agent, composed of $(num_q, rem(p_i))$

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 (the notion of equivalence is defined below). 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 (as described below), and then locally processed.

5.1.2 Equivalent Entries in the Agent Log Table

Obviously, one kind of equivalence is when the new record is completely identical to an existing entry, that is, they exactly match on all the three fields described above – in this case, the incoming agent is dropped.

There are more subtle equivalences, however, that arise when all the fields are the same including num_q except that there are differences in the $rem(p_i)$ value. This is shown in the following example: Consider the case where $rem(p_i)$ of the log entry is $\mathrm{L}{*}2 \cdot \mathrm{G}$ and that of the new entry is $\mathrm{L}{*}1 \cdot \mathrm{G}$ – in this case the $\mathrm{L}{*}2 \cdot \mathrm{G}$ is effectively a "superset" of $\mathrm{L}{*}1 \cdot \mathrm{G}$ in that it will have already covered all paths that $\mathrm{L}{*}1 \cdot \mathrm{G}$ would have taken. Therefore the new entry should not be considered. A more complex case is if the $rem(p_i)$ of the new entry has $\mathrm{L}{*}4 \cdot \mathrm{G}$ – here some of the paths have already been considered earlier (those corresponding to $\mathrm{L}{*}2 \cdot \mathrm{G}$) but not all – for example, the path $\mathrm{L} \cdot \mathrm{L} \cdot \mathrm{L} \cdot \mathrm{G}$ would not have been processed (assuming it exists). So, here we have a case of the new entry being a "superset" of the existing log entry and we have to ensure that *only the difference is processed*. For this, the query will have to be *rewritten*.

In general, let $(\mathrm{A}*\mathsf{m}) \cdot \mathrm{B}\ \ q_k \cdots$ be the query received by a site S through an agent with QueryID q, Entrypoint E, and remaining number of sub-queries t. Then, if a previous log entry of the form [E, q, (t, ($\mathrm{A}^*\mathsf{n} \cdot \mathrm{B}$))] exists, the following steps are taken based on the relative values of m and n :

$\mathbf{m} \leq \mathbf{n}$: Ignore the new entry and don't process the query further, i.e. purge the query.

$\mathbf{m} > \mathbf{n}$: Do the following :

1. Replace the existing log entry with the new entry; [E, q, (t, $(a * \mathsf{m}) \cdot b$)].
2. *Rewrite the query* stored in the agent, replacing $\mathrm{A}^*\mathsf{m} \cdot \mathrm{B}$ with
   $\mathrm{A} \cdot \mathrm{A}*(\mathsf{m}\text{-}1) \cdot \mathrm{B}$ and then undertake the standard agent processing.

Step 2 above implies that we are effectively forcing site S to function only as a PureRouter. Because, otherwise, if B included the null link, we would have evaluated the query at this site. It is easy to see that the agent will be rewritten at the first n sites it subsequently encounters. Therefore, it may appear that a more efficient solution would have been to rewrite the agent only once as $a^{n+1} \cdot a*(\mathsf{m} - \mathsf{n} - 1) \cdot \mathrm{B}$ where a^i denotes A concatenated with itself i times. This would indeed be correct in ensuring that this agent subsequently only chooses paths that have not already been taken – the problem, however, is that comparing and updating the log table entries at the downstream sites becomes ambiguous. For example, it would not be possible to distinguish between a "real" PRE that has $\mathrm{L} \cdot \mathrm{L}$ and a rewritten version of a PRE that originally had $\mathrm{L}*2$. To avoid this problem, we have chosen in DIASPORA to rewrite the query as often as required even if it were possible to rewrite it only once.

If no equivalence of the above forms can be established with any existing log entry, a new entry is inserted into the log table and the agent is subsequently processed in the normal manner.

To ensure that the log table does not take undue space, the old entries in the table are periodically purged. The periodicity of the purging is a configuration parameter that should be set based on the disk storage available and the processing duration of typical queries. Note that even if the purging time is incorrectly set too low resulting in duplicate queries being recomputed, it only affects the performance of the system but not the correctness of the results returned to the user.

Another point to note is that with the incorporation of the Agent Log Table, a minor modification has to be made to the CHT protocol discussed in the Section 4.4.1: A new entry that is equivalent to a previous entry should not be entered into the CHT since this new entry represents a duplicate agent that will be detected and dropped at the target site.

5.2 Reduction of Network Traffic

As discussed before, 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 [Mendelzon *et al.* 1997; Lakshmanan *et al.* 1996; Konopnicki and Shmueli 1995]. 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 web-site, 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 4.6, therefore not requiring additional termination messages from the query site to the sites currently hosting the agents of this query.

6 PERFORMANCE EVALUATION

In the previous sections, we discussed the design features of the DIASPORA system, and the associated optimizations. Based on this design, a prototype implementation of DIASPORA has been developed. The prototype is completely implemented in Java using JDK1.2 [Java 1997], with the parsers generated from the JavaCC [Javacc 1997] parser generator. Details of the implementation are available in [Ramanath 2000].

We evaluated our prototype of the DIASPORA system on a testbed of representative sites on our campus network. This set included, apart from the main IISc web-site, three departmental web-sites – Electrical Communication Engineering (ECE), Dept. of Metallurgy (MT), Supercomputer Education and Research Center (SERC) – and two lab web-sites – Database Systems Lab (DSL) and SERC Students Lab (SSL). The document characteristics of each of these web-sites, in terms of the number and total size of the documents hosted locally, is shown in Table 1.

We ran a variety of queries on the above test-bed and, due to space limitations, present here the results for only the example query presented earlier in Section 3.2. The remaining results are available in [Ramanath 2000]. The queries were submitted from a user-site that was also located on our campus network but *different* from the web-sites mentioned above.

6.1 Results for the Example Query

In the example query presented in Section 3.2, the search is limited by the traversal expressions in the WHERE clause. However, even though the search space has been narrowed down, a centralized query processing system

Site	No. of documents	total size (bytes)
IISc	94	556086
MT	166	775320
ECE	266	1312843
DSL	110	358262
SSL	241	335614
SERC	204	814961

Table 1: **Site Information**

(hereafter referred to as CENT), would be *completely outperformed* by DIASPORA. This is because CENT would need to import *all* documents in the traversal paths, whereas DIASPORA only requires the query to be sent to the site and the results shipped back. Hence, we undertook the following alternative assessment instead: Since the query makes use of keywords, we assumed that the documents containing these keywords in the appropriate scopes were "magically" known apriori, and only they would have to be imported and the result graph could be subsequently formed. Note that this is the *minimum* number of documents that would need to be imported by any CENT. In fact, this is a *conservative* assessment since we may also need to download some additional documents in order to form a fully connected result graph.

The query-processing starts from the main IISc web-site and agents travel to all of the remaining five sites, as follows: the query is forwarded from the IISc site to the MT, ECE and SERC sites and from the SERC site it is forwarded to the DSL and SSL sites.

6.1.1 Network Traffic Comparison

For the example query, the network traffic statistics of DIASPORA and CENT are shown in Table 2. The second column, which is for DIASPORA, reflects the cost of sending the query to the site as well as the results returned from that site. The third column indicates the number of queries forwarded from the site. The fourth and fifth columns, which relate to CENT, specify the number of documents in which at least one of the keywords was found and the total cost of importing all these documents, respectively. The last column shows the percentage of network traffic that is saved by DIASPORA's query-shipping approach as compared to CENT's data-shipping approach – these values indicate that DIASPORA significantly outperforms CENT, with traffic reductions well above the 50% mark. Note that the statistics for DIASPORA include the cost of forwarding the queries to other sites as well as the cost of returning the results back to the user (the SSL site returns an empty result with only CHT information since it does not host any document that satisfies the query).

1	2	3	4	5	6
	DIASPORA		CENT		
Site	**total size of results+query (bytes)**	**no. of forwarded queries**	**no. documents containing results**	**size of documents containing results (bytes)**	**percentage savings**
IISc site	4026	3	2	16822	76
MT	32512	0	28	141300	77
ECE	7523	0	9	22268	66
DSL	3179	0	4	7989	60
SSL	185	0	0	0	–
SERC	28599	2	14	154047	81
Total	76024	5	57	342426	78

Table 2: **Network Statistics for the Example Query**

6.1.2 Response Time Comparison

1	2	3	4	5	6
	DIASPORA		CENT		
Site	**local query processing time**	**cumulative response time**	**document download time**	**db+result construction time**	**cumulative response time**
IISc	82	174	110	2072	2182
MT	200	980	1570	4901	8653
ECE	4238	4453	300	8388	10870
SERC	149	483	590	2193	4965
DSL	285	800	40	1408	6413
SSL	161	660	0	0	4965
Total	5115	–	2610	18962	-

Table 3: **Response Times for the Example Query**

Turning our attention to the user response times, these statistics for DIASPORA and CENT are presented in Table 3 (all times are in milliseconds). The second column indicates, for DIASPORA, the local query processing time at each site. The third column indicates the *cumulative* response time, that is, how much time did it take for the result from each query-site to reach the user-site since the time the user originally submitted the query. For example, it took only 483ms after the submission of the query for the results from the SERC site to be available at the user-site. Similarly, results from the SSL site reached the user-site 660ms after submission of the query. Thus, the *overall*

response time to receive the entire set of results for the query is 4453ms (the response time of the ECE site).

The response times for CENT were measured for the same traversal path as that followed by DIASPORA. That is, CENT accesses and processes the query in parallel for the data from the MT, ECE and SERC sites, and after the SERC data has been processed, the data for the DSL and SSL sites is subsequently accessed and processed. The corresponding results are shown in columns 4, 5 and 6 of Table 3. We observe here that both the individual as well as the total response times of CENT are substantially higher than that of DIASPORA. Overall, DIASPORA is nearly twice as fast as CENT (compare the response times in columns 3 and 6 for ECE), and for the other sites, the speed improvement is by almost an *order of magnitude*.

The above results served to highlight the performance benefits achievable from the DIASPORA query processing protocol as compared to a centralized approach. Apart from these experiments, we also assessed the impact of the design choices in DIASPORA – in particular, the impact of the CHT protocol . We found that the overhead of using this protocol hardly impacts on the efficiency of the system. Even in the worst case scenario, where in the "critical path" (the longest path taken by the query) of the query, all intermediate sites are PureRouters and only the last site in the path returns results, the overhead of the CHT protocol was about 1%. More details of these experiments are available in [Ramanath 2000].

The above results indicate that when all the relevant data is stored remotely, DIASPORA is clearly the system of choice. In our current work, we are also looking into how it can be integrated, and its performance further enhanced, with the use of *web-caches* at both the user-site and the query-sites.

7 RELATED WORK

Web data is an example of "semi-structured" data [Abiteboul 1997], an area that has seen much research activity in recent times. As mentioned in the Introduction, the main challenges in the development of a web database include the following: (i) developing a suitable data model for web data and "wrappers" for wrapping the web data so that it conforms to the required data model, (ii) developing suitable query languages to query the web database, and (iii) query processing and optimization.

Semi-structured data has been studied in the context of data integration systems (for example, Tsimmis [Garcia-Molina *et al.* 1995]). Data models for semi-structured data have been proposed in [Garcia-Molina *et al.* 1995] and [Buneman *et al.* 1996]. Research has also gone into generation of wrappers and a considerable amount of literature is available (for example, [Hammer *et al.* 1997; Ashish and Knoblock 1997; Adelberg 1998; Grumbach and Mecca 1999]. Query languages for semi-structured data such as Lorel [Abiteboul *et al.* 1997], UnQL [Buneman *et al.* 1996] and StruQL [Fernandez *et al.* 1998] (in the context of web-site management) have also been proposed.

While there are a variety of interesting design proposals for web database systems, such as W3QS [Konopnicki

and Shmueli 1995], WebSQL [Mendelzon *et al.* 1997], WebLog [Lakshmanan *et al.* 1996], WHOWEDA [Bhowmick *et al.* 1998; Bhowmick *et al.* 2000], Araneus [Atzeni *et al.* 1997], WebOQL [Arocena and Mendelzon 1998], etc., for lack of space we review below the salient features of only two systems WHOWEDA and WebOQL. For a more comprehensive overview on the integration of web and database technology, please refer to [Florescu *et al.* 1998].

WHOWEDA (*W*are*h*ouse *o*f *We*b *Da*ta) is a system built around the Web Information Coupling Model platform, which incorporates a node/link representation of the web – a node corresponds to a document and a link corresponds to the hyperlink between two documents. A collection of node and link objects constitute a "web-tuple", which therefore represents a set of directed graphs. A "web-table" is then defined as a set of web-tuples along with a "web-schema" which describes the web-table. Finally, a "web algebra" is defined that supports operations on web-tuples stored in web-tables. The operators in the web algebra include "web select", "web join", "web intersection", global/local "web coupling", etc. Given a data warehouse built with the above framework, the user can extract information by using a *query graph* which is a directed graph containing nodes and links. Each node/link in the query graph may have complex constraints imposed on them. While both WHOWEDA and DIASPORA try to relate keywords across documents, at a more detailed level, there are some differences: First, WHOWEDA employs a node/link model whereas we use an edge-labelled graph. Second, our modeling extends to document internals also.

WebOQL is a language designed for restructuring trees. A data structure named *hypertree* is utilized to model the document (for example, an abstract syntax tree of an HTML document is the hypertree for that document). A collection of hypertrees forms a *Web*. WebOQL now operates on hypertrees to extract arbitrary trees and to restructure one hypertree to another. There are two primary differences between our approach and WebOQL. The first is in the modelling of the document. Though in both cases the document model is automatic in construction, our model infers semantic information in the document, whereas there no such attempt is made in the data model of WebOQL. The second difference is that we rely only on keywords to extract information, whereas WebOQL requires more precise knowledge of the format of the data stored in hypertrees to be fully effective. For example, the tag type in which the title of publications are present would help in extracting them and similar information would help in extracting and restructuring the hypertree. This is a potential limitation since different documents might use different tags to express the same data.

For all the systems mentioned above, the query processing is centralized which is in sharp contrast to our distributed approach. It should be noted, however, that DIASPORA's query processing mechanism can be integrated with most of these systems. We now briefly mention the few algorithms that have been proposed, in parallel with our work, for distributed query processing. An algorithm for distributed processing of query paths using asynchronous message passing is presented in [Abiteboul and Vianu 1997]. The query path is successively shortened as and when a part of it is satisfied by a site. The remaining part of the query is sent to successive sites. This approach appears similar to ours, but their termination detection is quite different: A message is sent along with the query from the user

site to the StartPoint and when the StartPoint acknowledges this message, it indicates the termination of the query. The StartPoint acknowledges the message only if all the sites to which it had forwarded the query have acknowledged the message. This propagates to further sites until a site answers the query.

In contrast to the above, [Suciu 1997] describes how a query can be decomposed into several components and each component sent to a different site which return results. The results are then recombined at the original site. The algorithm presented here requires that the sites involved in processing the query are known in advance and that "input nodes" (all local documents which are referenced from documents from other sites) are also known in advance.

Both the above papers focus primarily on the theoretical aspects and do not describe any implementation mechanisms for their models. A distributed query processing algorithm where agents called "navigators" are dispatched to various web-sites to find "qualified paths" for a given PRE was described in [Katoh *et al.* 1998]. In this algorithm, an automaton is first constructed for a given PRE. This automaton is then broken down into sub-automatons each of which may be dispatched to different sites to determine if nodes which satisfy the PRE of the sub-automatons exist. The main difference between this approach and ours is that their navigators are *co-ordinated centrally* by the user-site whereas no such co-ordination is required in our approach.

8 CONCLUSIONS

In this paper, we have described DIASPORA, a new querying system intended for use in Web subnets. It features a graph-based data model that represents the relationships of data elements within Web documents, infers semantic meta-data information from both markup tags and element values, and is fully automatic in its construction. The query language for operating on this model supports both content and structural queries, and also allows users to specify scopes for searching and traversal of the Web. Results are returned as a set of graphs and are processed to show a connected graph that places the keywords given by the user in context so that it is easy to determine the relevance of each result. Overall, the model and the query language integrate some of the ideas previously proposed in the literature and also incorporate additional new features.

The most novel feature of DIASPORA's design is its distributed query processing system. User queries are decomposed into an equivalent set of sub-queries which are forwarded from site to site using a socket communication platform, with the results computed at each query-site directly returned to the user-site. The system has been designed so as to not require a central controlling authority, thereby allowing the query forwarding and processing to be highly distributed. Further, a variety of novel issues, not typically encountered in the traditional distributed database system context, have been addressed – these include determining query completion, handling query rewriting, supporting query termination, returning results and preventing multiple computations of a query at a site due to the query arriving at the site in different paths in the hyperlink framework.

A Java-based implementation of DIASPORA is currently operational and initial tests of this system on our campus network show that it considerably reduces network traffic and improves user response times as compared to equivalent centralized systems. In fact, this improvement holds even under the extreme and infeasible assumption that the centralized system *apriori* knows the identities of all the remote documents containing results.

In summary, we expect that the DIASPORA system will be of use in a variety of web-related applications, including development of search-engine indices and sitemaps, apart from answering ad-hoc user queries that relate to both the content and the link structure of Web documents. Moreover, we expect the utility of its distributed processing feature to increase even further with the advent of XML documents, which support fine-grained querying, especially when these documents are hosted on backend database engines. DIASPORA also opens up opportunities for mining user queries to improve commercial and public services offered by web-sites.

REFERENCES

Abiteboul, S. (1997), "Querying Semi-Structured Data," In *Proceedings of the International Conference on Database Theory*, pp. 1–18.

Abiteboul, S., D. Quass, J. McHugh, J. Widom, and J. Weiner (1997), "The Lorel Query Language for Semistructured Data," *International Journal on Digital Libraries 1*, 1, 68–88.

Abiteboul, S. and V. Vianu (1997), "Regular Path Queries with Constraints," In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 122–133.

Adelberg, B. (1998), "NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents," In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 283–294.

Arocena, G. and A. Mendelzon (1998), "WebOQL: Restructuring Documents, Databases and Webs," In *Proceedings of the 14th International Conference on Data Engineering*, pp. 24–33.

Ashish, N. and C. Knoblock (1997), "Wrapper Generation for Semi-structured Internet Sources," *SIGMOD Record 26*, 4, 8–15.

Atzeni, P., G. Mecca, and P. Merialdo (1997), "To Weave the Web," In *Proceedings of the 23rd Very Large Data Bases Conference*, pp. 206–215.

Bhowmick, S., S. Madria, W.-K. Ng, and E.-P. Lim (2000), "Detecting and Representing Relevant Web Deltas using Web Join," In *Proceedings of the 20th International Conference on Distributed Computing Systems*.

Bhowmick, S., S. K. Madria, W.-K. Ng, and E.-P. Lim (1998), "Web Warehousing System: Design and Issues," In *Proceedings of the International Workshop on Data Warehousing and Data Mining*, pp. 93–104.

Buneman, P., S. Davidson, G. Hillebrand, and D.Suciu (1996), "A Query Language and Optimization Techniques for Unstructured Data," In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 505–516.

Deutsch, A., M. Fernandez, and D. Suciu (1999), "Storing semistructured data with STORED," In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 431–442.

Fernandez, M., D. Florescu, J. Kang, A. Levy, and D. Suciu (1998), "Catching the Boat with Strudel: Experiences with a Web-site Management System," In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 414–425.

Florescu, D., A. Levy, and A. Mendelzon (1998), "Database Techniques for the World Wide Web: A Survey," *SIGMOD Record 27*, 3, 59–74.

Fujimoto, R. (1990), "Parallel Discrete-Event Simulation," *Communications of the ACM 33*, 10, 30–53.

Garcia-Molina, H., J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom (1995), "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS," In *Proceedings of the AAAI Symposium on Information Gathering*, pp. 61–64.

Grumbach, S. and G. Mecca (1999), "In Search of the Lost Schema," In *Proceedings of the International Conference on Database Theory*, pp. 314–331.

Hammer, J., H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo (1997), "Extracting Semistructured Information from the Web," In *Proceedings of the Workshop on Management of Semistructured Data*, pp. 18–25.

Java (1997), "Java 2 SDK, Standard Edition," http://java.sun.com/products/jdk/1.2/.

Javacc (1997), "Java Compiler Compiler, (JavaCC) Version 1.0)," http://www.metamata.com.

Katoh, K., A. Morishima, and H. Kitagawa (1998), "Navigator-based Query Processing in the World Wide Web Wrapper," In *Proceedings of the 5th International Conference of Foundations of Data Organization*, pp. 191–199.

Konopnicki, D. and O. Shmueli (1995), "W3QS: A Query System for the World-Wide Web," In *Proceedings of the 21st Very Large Data Bases Conference*, pp. 54–65.

Lakshmanan, L., F. Sadri, and I. Subramanian (1996), "A Declarative Language for Querying and Restructuring the Web," In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering*, pp. 12–21.

Litzkow, M., M. Livny, and M. W. Mutka (1988), "Condor - A Hunter of Idle Workstations," In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pp. 104–111.

Mendelzon, A., G. Mihaila, and T. Milo (1997), "Querying the World Wide Web," *International Journal on Digital Libraries 1*, 1, 54–67.

Milojicic, D., W. LaForge, and D. Chauhan (1998), "Mobile Objects and Agents (MOA)," In *Proceedings of the USENIX Conference on Object-oriented Technologies and Systems*, pp. 1–14.

Nguyen, T. and V. Srinivasan (1996), "Accessing Relational Databases from the World Wide Web," In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 529–540.

Raggett, D. (1997), "HTML 3.2 Reference Specification," http://www.w3.org/TR/REC-html32.html.

Ramanath, M. (2000), "DIASPORA: A Fully Distributed Web-Query Processing System," Master's thesis, Indian

Institute of Science.

Shanmugasundaram, J., H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. F. Naughton (1999), "Relational Databases for Querying XML Documents: Limitations and Opportunities," In *Proceedings of the 25th Very Large Data Bases Conference*, pp. 302–314.

Suciu, D. (1997), "Distributed Query Evaluation on Semistructured Data," http://www.research.att.com/∼suciu/strudel/external/files/_F6

XML (1998), "Extensible Markup Language (XML) 1.0," http://www.w3.org/XML.