THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

HYBRID KEYWORD SEARCH ACROSS PEER-TO-PEER FEDERATED DATA

By

JUNGKEE KIM

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Spring Semester, 2005

The members of the Committee approve the dissertation of Jungkee Kim defended on April 6, 2005.

Gregory Riccardi
Professor Co-Directing Dissertation

Geoffrey C. Fox
Professor Co-Directing Dissertation

Lawrence Dennis
Outside Committee Member

Gordon Erlebacher
Committee Member

David Whalley
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

I would like to dedicate this dissertation to my parents, to my wife Hiejou, and to my son Eric Namyoon. I would also like to thank Dr. Fox, Dr. Riccardi, Dr. Erlebacher, Dr. Whalley, and Dr. Dennis for every concern. Especially I thank Bryan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Internet provides a general communication environment for distributed resource sharing. XML has become a key technology for information representation and exchange on the Internet, increasing the opportunity for integration of the various data formats. The *World Wide Web* (WWW) is the example par excellence of a document-based distributed system on the Internet. As the size of the Web has increased, various problems with looking up a resource location on the Internet have emerged. Web search engines provide clues for resource location, but they have no semantic schema and often produce meaningless keyword search results. The *Semantic Web* suggests an alternative solution for the semantic problem on the Web. It provides multiple relation links with directed labeled graphs, and machines like Web crawlers can understand the relationship between different resources. But due to the need for sophisticated domain description and lack of unified definitions, many Web pages are not part of the Semantic Web. Meanwhile, recent public attention to *peer-to-peer* (P2P) networks has stimulated research on overlay P2P networks on top of the Internet. Those studies open possibilities for another form of distributed resource sharing on the Internet.

In this dissertation we describe the design of a hybrid search that combines metadata search with a traditional keyword search over unstructured context data. This hybrid search paradigm provides the inquirer additional options to narrow the search with some semantic aspects through the XML metadata query. We tackle the scalability limitations of a single-machine implementation by adopting a distributed architecture. This scalable hybrid search provides a total query result from the collection of individual inquiries against independent data fragments distributed in a computer cluster. We demonstrate our architecture extends the scalability of a native XML query limited in a single machine and improves the performance of queries. Finally we generalize our hybrid architecture to more scalable searches over a P2P overlay network. This generalization may give an intermediate search paradigm on the Internet—providing semantic value through XML metadata that are simpler than those of the Semantic Web.

CHAPTER 1

INTRODUCTION

1.1 Research Objectives

Ever since the Internet was introduced as a communication and resource sharing environment, there have been efforts to utilize its enormous and rapidly proliferating resources. There are two archetypal approaches for search on the Internet. One is searching over structured data, and the other is searching over unstructured data. A relational database is representative of structured data, while information retrieval represents search over the unstructured data.

A Web search engine is a typical example of search on the Internet. Its technologies are rooted in information retrieval. In 2005, Google Search [1]—the most famous of search engines—reaches more than 8 billion Web pages and provides very fast information retrieval through its collected indexes. However search engines cannot visit every connected resource. Furthermore the search results of the Web search engines are often highly irrelevant, because the crawled Web contents are indexed text without proper semantic schema. Lucky combination of keywords may be needed to find the target information.

The *Semantic Web* [2, 3] is a superlative extension of the Web. It also includes multiple relation links with directed labeled graphs by which machines like Web crawlers can interpret the relationship between resources. Meanwhile the ordinary Web has a single relationship and a machine cannot infer further meaning. To represent the relations of the objects on the Web, the object terms should be defined under a specific domain description—an ontology. Domain experts are usually needed to design an ontology due to the sophisticated definition required. Currently, many Web pages include no such semantic content, and no unified definition of general semantic agreement exists.

Keyword search in databases [4, 5] is one response to the new Internet environment from the traditional database society. Both Web integration of legacy database management systems, and dynamic Web publication through embedded databases, strongly benefit from keyword search capability on the databases. This is because conventional query on databases requires knowledge of the schema to extract the target information. Additionally recent assimilation of semistructured schema to the Web [6, 7] and databases makes it more difficult for ordinary users to utilize the proper inquiry. Though the keyword based search simplifies the search on the database, it loses the inherent meaning of the schema. So keyword search on databases, like Web search engines, does not return results based on semantic criteria.

We propose a hybrid keyword search that combines metadata search with a traditional keyword search over unstructured context data. This search paradigm lies between the two search approaches described above. The hybrid search is fundamentally based on metadata attached to each unstructured document. We adopt XML—the de facto standard format for information exchange between machines—as a metalanguage for the metadata. Attaching metadata to legacy information systems based on keyword-only search enables narrowing the search category. We will first demonstrate our hybrid search on a local machine; but we expose the scalability limitations in a single-machine implementation, as well as some query performance inefficiency under specific environments. Particularly, the native XML database we used had very limited scalability. So we subsequently adopt a distributed strategy to improve the scalability of our hybrid keyword search.

Meanwhile, the resurrection of the *Peer-to-Peer* (P2P) based network makes it possible to provide customized overlay networks on top of the Internet. The desire for sharing resources such as music files quickly led to creation of a number of peer groups [8]. The idle CPU time of home computers can be exploited in a large computation, and for example contribute to finding intelligent life in outer space [9]. We have focused on a mutual interest group whose resources need to be shared and searched through an overlay network. The peer-to-peer connection of the search services provides low cost scalability. Associating additional schema to the resource is not as difficult as for the Semantic Web, but it will increase the information transparency. This model could be extended to give a simplified compromise between the Semantic Web and the Web search engine. Another point for the peer-to-peer overlay network is that it increases network traffic. Modern network technology overcomes the barrier of text

only communications, and large amounts of multimedia content are delivered through the current Internet. We generalize our hybrid architecture to greater scalability over an open peer-to-peer framework (JXTA [10]). This architecture may have a practical bridging role for information search—providing semantic value through metadata whose implementations are much simpler than those of the Semantic Web.

1.2 Outline of this Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we present an overview of metadata standards such as XML, DTD, XML schema, RDF, XPath, XPointer, and XLink. In Chapter 3, we present database technologies for XML. In Chapter 4, we present peer-to-peer technologies and message-oriented communication. Chapter 5 describes and evaluates the hybrid search and its repository for the metadata and the unstructured data targeted keyword search. Chapter 6 presents and evaluates our search architecture on distributed databases. Chapter 7 proposes the hybrid P2P keyword search and evaluates its P2P communications. Related work is described in Chapter 8. We conclude and outline possibilities for future research in Chapter 9.

CHAPTER 2

METADATA

Since the Extensible Markup Language (XML) [11] was introduced as a standard document form, the use of XML as the data exchange format between applications has dramatically increased. Before the appearance of XML, application dependent data formats were used for data exchange. However, the World Wide Web environment is so huge that application dependent data formats require a lot of coding effort for synchronizing data between different applications, due to the many different data formats on the Web. Though Hyper Text Markup Language (HTML) protocol [12] is used on the Web, it only describes how the data are shown through Web browsers with the fixed tag format. XML provides user-defined tags, and is simple to produce and flexible. With nested tags, XML can represent not only a hierarchical tree structure, but also a graph structure using special attributes. Some applications may need an agreement on the XML document format, because such standardization would reduce the errors and increase the efficiency in exchanging data. Document Type Definition (DTD) and XML Schema [13] are used to force the production of XML documents satisfying particular rules.

To extract key information through the Web, metadata can be used to designate the desired information. The Resource Description Framework (RDF) [14] provides a description of metadata mainly relating to data on the Web. Through RDF, the semantic meaning of the Web resources can be revealed to machine-oriented systems, for example, search engines. The RDF Schema [15] is defined to constrain the vocabulary of RDF documents. From the extensions of RDF and RDF Schema, a semantic markup language can be generated for intelligent semantic descriptions of Web documents. To support linking for XML documents, XLink [16], XPointer [17], and XPath [18, 19] were introduced by the World Wide Web Consortium.

2.1 XML

The Extensible Markup Language (XML) [11] is a standard document specification, a subset of the Standard Generalized Markup Language (SGML) format [20]. Though SGML is powerful enough to have been used by the U.S. government and publishing companies for making documents, its implementation is considered to be difficult and complex. The Hypertext Markup Language (HTML) [12] is another application of SGML, but HTML only presents the shape of the documents on the Web. The tags of HTML are fixed and the HTML has no mechanism for data validation. Those features limit HTML to Web information manipulation, and prompted to the emergence of XML.

XML consists of a well-formed structure, rooted in a prolog, one or more elements including balanced start- and end-tags with attributes, and miscellaneous optional features including comments, processing instructions, and white space. A typical example of the prolog is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

This indicates that this is an XML document and the version number is "1.0." The other *attributes*, "encoding" and "standalone," are optional. The encoding denotes that this document used Unicode Transformation Format 8-bit (UTF-8) encoding. As XML is designed to support the International code, the encoding names can be whatever the parser supports, but the name is recommended to be registered with Internet Assigned Numbers Authority. The standalone document declaration is also an optional attribute and it indicates whether the XML document is affected by an external markup declaration like the DTD or XML schemas.

An *element* forms a root of a hierarchical tree structure for an XML document. Other elements can be added and should be nested within each other. If there is no content in an element, the empty-element tag, *''<TAG-NAME/>''*, can be used instead of a pair of tags. Tags are user-defined and that feature makes XML very different from HTML. Attributes are used to attach additional information of an element. They are located in start-tags or empty-element tags. Through attributes of type ID, IDREF or IDREFS [21], XML elements can be uniquely identified and form links. This linking mechanism allows XML to represent

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<!-- This is an example car.  -->
<car id="J544XD" state="NY">
   <company> Toyota </company>
   <model> Corolla </model>
   <type> DX </type>
   <year> 1996 </year>
   <color> white </color>
</car>
```

Figure 2.1. A car description in XML

graph-structured information as well as tree-structured. In special CDATA sections, any markup data are is interpreted as text data.

Figure 2.1 shows an example of an XML document. The first line is the prolog. The version attribute should be declared. The optional attributes *encoding* and *standalone* show that the code for this XML document is "UTF-8" without any schema ("standalone" is "yes"). The second line is empty—XML allows empty lines for good formatting. The third line is masked as a comment, which does not present any meaning in the XML structure. The comment only provides optional information to readers. There is only one root element named "car" in the document and other elements are nested in the root element. The car element has two attributes. The *id* attribute identifies a specific car and the *state* attribute means the registered state of the car. Each element opens with start-tag *''<TAG-NAME>''* and closes with end-tag *''</TAG-NAME>''*, and there are no overlapping tags. So this XML example is said to be *well-formed*.

2.2 DTD and XML Schema

Document Type Definition (DTD) and XML Schema are ways to define the structure of XML documents. DTD has been used in SGML for over twenty years and XML Schema is newly specified by the World Wide Web Consortium (W3C) [13]. The goal is to make rules to construct XML documents. For many purposes, user-defined tags alone don't provide a sufficiently rigorous structure for the XML information exchanges. By requiring the same

DTD or XML Schema, two different applications can agree on a particular structure for an XML document. If a well-formed XML document satisfies a DTD or XML Schema, the document is said to be *valid*.

The XML Schema specification reflects the demands of users, who have found DTD too limited. The schema has many improved features over DTD. Before defining XML Schema, there were several attempts to improve the functionality of the schema language for XML documents; some examples are Document Definition Markup Language (DDML), Document Content Description (DCD), Schema for Object-Oriented XML (SOX), and Microsoft's XML-Data for BizTalk. The W3C consortium activity for XML Schema considered those schemas in producing their own design.

2.2.1 DTD

DTD format is very different from XML. A DTD is usually included in the prolog part of an XML document using the "!DOCTYPE" tag. The DTD can be defined externally in a separate file, designated with a file name or a Uniform Resource Identifier (URI). The typical blocks of a DTD are elements and attributes. BNF syntax is as follows:

```
<!ELEMENT> <element-name> <element-type>
<!ATTLIST> <attribute-name> <attribute-type> <attribute-option>
```

Figure 2.2 shows an example DTD for the car document of Figure 2.1. In the example, the "car" element is a *non-terminal* and the other elements are *terminals*. The non-terminal element, "car," has five sub-elements: company, model, type, year, and color in that order. It is called a *sequence*, which restricts the order of sub-elements present. *Choice* is another group option for the sub-elements and it gives a list of alternatives for them. The vertical bar ("|") is used as the delimiter for choices and the comma for sequences.

In the sequence of the example, all but the type element will appear exactly once. The type element can be included optionally. This is indicated by the suffix, "?." Other allowed suffixes include "+," which means one or more elements can appear, and "*," which means zero or more can appear.

"#PCDATA" in terminal elements stands for parsed character data, which denotes text that has no markup. That is the only way to represent text in DTD, and this was one of

```
<!ELEMENT car(company, model, type?, year, color)>
<!ATTLIST car
          id CDATA #REQUIRED
          state CDATA #IMPLIED>
<!ELEMENT company (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT color (#PCDATA)>
```

Figure 2.2. DTD for a car document

motivations for the invention of XML Schema. The element content can also be *empty* or *any*. The *empty* element has no content but may have attributes. The *any* element has no restriction for that element.

The "car" element in the example has two attributes, which are declared in the DTD. The order of the attributes is not constrained. Both of the attributes have the same data type, character data (CDATA). Other attribute types like ID, IDREF, and IDREFS are also very useful and have key roles in representing graph structures in XML format. The final term in an attribute description specifies whether this attribute is optional or required. The option for the attribute "id" is "#REQUIRED," and this attribute must appear in the every "car" element. The "#IMPLIED" option in "state" means the attribute can be optional. Other option is "#FIXED" and this type attribute should have a default value. The fixed value cannot be changed by the user.

2.2.2 XML Schema

While DTD syntax is comparable to Extended Backus Naur Form (EBNF) [22], XML Schema uses XML document syntax. By supporting *namespaces*, XML Schema allows several sources of document definitions to be used in a single document. In DTD, a new DTD is needed to combine multiple DTDs. The XML Schema supports 44 datatypes including *string, decimal, time* and *date*, whereas DTD only provides 9 XML-related primitive types.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="car" type="CarType"/>

<xsd:complexType name="CarType">
  <xsd:sequence>
    <xsd:element name="company" type="xsd:string"/>
    <xsd:element name="model" type="xsd:string"/>
    <xsd:element name="type" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="year" type="xsd:decimal"/>
    <xsd:element name="color" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="state" type="xsd:string"/>
</xsd:complexType>

</xsd:schema>
```

Figure 2.3. XML Schema for a car document

Inheritance is another major feature of XML Schema, which is not present in DTD. This allows reusing existing structures by extending or restricting the base types.

Figure 2.3 includes an example of XML Schema, which reproduces the schema presented in Figure 2.2 in DTD format. The *car* schema has one *schema* element with sub-elements *element* and *complexType*. In the schema element, a namespace has been declared. The prefix "xsd:" associated with the namespace is used on each of the elements. The prefix name of a namespace can be an arbitrary value and different namespaces from different sources can be used. In the multiple namespaces, the different prefix names specify the meanings of elements and attributes, which are followed by the prefix. In this example, the association forces the elements and simple types to be identified with the XML Schema language. In XML Schema, elements may have simple types or complex types. A simple type does not include elements. Many simple types, such as "xsd:string," are defined in the XML Schema. A complex type may have nested elements and carry attributes optionally. The type of the

car element is defined as a complex type, *CarType*, in the example. As in the DTD example, the elements of a complex type can be ordered with a *sequence* tag. All the nested elements except the *year* element have a *string* type. This is declared in the *type* attribute. The *decimal* type in the year element is a number. If the year needs to be restricted to four-digit numbers, another simple type, *gYear*—pre-defined in XML Schema—can be used instead as follows:

```
<xsd:element name="year" type="xsd:gYear"/>
```

If the *state* attribute has to be two capital letters, a new type can be defined as follow:

```
<xsd:simpleType name="StateType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

The equivalent of optional elements in DTD can be expressed in XML Schema using "minOccur" and "maxOccurs" attributes. Beyond the three restrictions of DTD (*, +, and ?), XML Schema can designate any number of minimum and maximum occurrences—for example, between 15 and 30. Any given attribute may appear at most once. The "use" attribute designates the attribute usage—one of *required*, *optional*, and *prohibited*.

The *fixed* attribute is used when the allowed value of an element or attribute is unique. For example, I may add another attribute *country* in the "car" element. The "state name" attribute is used in United States and the fixed value for the country could be "US." The definition of the attribute is as follows:

```
<xsd:attribute name="country" type="xsd:string" fixed="US"/>
```

2.3 Metadata

There have been many efforts to extract useful information through the Web. Especially, search engines devised various technologies to find the exact location of the desired information. But in practice many of them also show useless information. Moreover, they are

reported to reach less than one percent of the whole Web. That is because the scale of the Web is so huge, and keyword spamming is very widespread. Though some Web directory sites categorize the information manually, this is not for the machine-oriented system.

Metadata is "structured data about data." This could include catalogs of libraries, author lists of books, ranking of Web pages by frequency of reference, or the relations between indexes. Both human and machine generated information can be metadata.

2.3.1 RDF

The Resource Description Framework (RDF) is a W3C recommendation for a standard representation of metadata [14]. This framework is described in XML format. RDF is also defined by directed graphs. RDF has an innate function for machine-oriented data exchange between applications because of its XML features. XML and RDF provide semantic interoperability in the current Web domain, but XML only describes the document structure. RDF emphasizes semantic meaning of the Web resources by adding a data model for knowledge representation.

The basic block of RDF consists of three object types—*resources*, *properties*, and *statements*. A *resource* is anything that can be written as a Uniform Resource Identifier (URI) in the RDF expression. It can be not only a Web page but also an XML element. Anything with a URI could be a resource. A *property* is a specific characteristic, attribute, or relation of the resource—for example, "owner." Each property has a specific meaning, which can be classified by a schema related to the name of the property. A *statement* is a combination of a *resource*, a *property*, and a *value*. The parts of the statement are also known as the *subject*, the *predicate*, and the *object*. The object can be another resource or a literal, which might be any string or XML. Figure 2.4 presents an example of an RDF graph for the Web page of Florida State University.

In the figure, the oval shape node denotes a subject, the arc denotes a named property, and the rectangular shape is a node, which represents a literal. The graph represents the following statement:

```
"Florida State University is the owner of the resource http://www.fsu.edu."
```

It also can be read as:

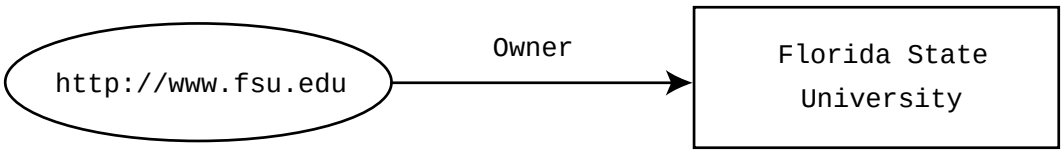

Figure 2.4. An example of RDF graph

```
"http://www.fsu.edu/ has owner Florida State University."
```

The statement can be written in the XML format:

```
<?xml version="1.0">
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-synfax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.fsu.edu/">
    <s:owner>Florida State University</s:owner>
  </rdf:Description>
</rdf:RDF>
```

The RDF XML syntax has a root element, *"<RDF>"*, but this element is optional when the *description* is known to be RDF from the application content. In RDF element, the namespace attributes designate the location of the declarations of the RDF elements with the prefix "rdf:", and the location of the schema declaration associating with the prefix "s:". The namespace declaration can alternatively appear in a specific *description* element, or even in property elements. The *description* element has the subject; the child elements describe the properties and the objects. In this example, "<s:owner>" and "</s:owner>"—a pair of tags—show the property. The object is "Florida State University."

Like the XML Schema for an XML document, an RDF Schema provides a vocabulary constraint facility for RDF documents. In RDF Schema, the classes of the resources are defined. The classes have the same role as in the object-oriented programming models. The classes have hierarchical structures and they are extended by subclass refinement. Terms including "Class," "subPropertyOf," and "subClassOf" are used in the basic type system

of RDF to define such classes. The reusability of metadata can be increased by using class concepts , because sharing schemas and adding subclasses to the existing schemas will be sufficient in many schema specifications.

Ontologies have a central role in the "Semantic Web"—a machine-understandable Web with intelligent services. They provide shared and precisely defined terms in a particular domain, for communications between human users and application systems. DAML+OIL—combines terminology from the older DARPA Agent Markup Language (DAML) and Ontology Inference Layer (OIL)—is an ontology language submitted to W3C as a semantic markup language for Web resources [23]. It extends RDF and RDF schema. The object-oriented structure of domains in DAML+OIL uses the terms "Class" and "Property." The usage of the term "class" is similar to that of RDF schema, but the classes of DAML+OIL are less restricted. For example, "subClassOf" class elements of DAML+OIL allow cyclic subclass-relations. A "property" is a binary relation, which defines the relation between two items. All expressions of description logic can be written in DAML+OIL terms. The DAML+OIL properties are of two types: "objectProperty" for object relations and "datatypeProperty" for datatype values. Those properties are ranged, and multiple ranges can apply to a property *conjunctively*. For example, positive integer range and numbers greater than 1 range will produce a property that has positive integers greater than 1. The restriction of domains in DAML+OIL is global, while the RDF Schema only has a single range and a local scope on domains. However, efficient tools for reasoning about Web resources and complete algorithms for the full DAML+OIL language are not provided yet [24].

Since the DAML+OIL language was submitted as part of a W3C recommendation, improved versions have been subsumed under the *OWL* Web Ontology Language [25]. OWL improves compatibility to other W3C recommendations and uses more meaningful names—for example, "daml:hasClass" has been changed to "owl:someValuesFrom". OWL provides three different versions—*OWL Lite*, *OWL DL*, and *OWL Full*. OWL Lite is a simplified subset of the language for the users who do not need full language features. OWL DL has all the constructs of OWL with some restrictions for the reasoning support. DL comes from *Description Logic*. OWL Full is just a terminology for the usage of OWL without further restrictions.

From the viewpoint of semantic interoperability, RDF is better than plain XML. RDF vocabularies are simple enough to manipulate huge volumes of data. Meanwhile, XML often regards ordered elements as important and has complex structure. Those features make it difficult for XML to handle large amounts of data. Additional data conversion is not necessary in RDF because RDF presents domain models naturally by defining objects and relations. Another benefit of RDF is independency of XML. In an XML document, a schema change may cause invalidity for a query based on the old structure of the XML document. RDF presents a semantic tree that is parsed with a usable set of triples, and the uninterpreted data is ignored [26].

2.3.2 XPath, XPointer, and XLink

As in hyperlinks of HTML documents, XML documents can be linked to other XML documents by using XML Linking Language (XLink) [16], XML Pointer Language (XPointer) [17], and XML Path Language (XPath) [18, 19]. XLink describes links between resources. XPointer points the reference through URI. XPath presents the location of specific parts of an XML document.

2.3.2.1 XPath

The main purpose of XPath is to designate parts of an XML document. XPath provides an extended addressing syntax defining a compact notation for node location in the XML document tree. XPath does not use XML syntax but is a string-based language. Extensible Stylesheet Language Transformations (XSLT) [27] and XPointer use the functionality of XPath.

2.3.2.2 XPointer

XPointer is used to identify specific fragments in XML documents via a URI. XPointer may select on the basis of XML ID attributes, or nodes in the hierarchical structure of an XML document selected using XPath. An XPointer can also reference an arbitrary user designation of a specific *point* or *range*—it doesn't have to be an XML node. A *range* can be specified with two *points*, as:

```
xpointer(id("start")/range-to(id("end")))
```

This XPointer locates the range between the start point of the element with ID "start" and the end point of the element with ID "end". XPointer is sufficiently complex to handle most usage cases.

2.3.2.3 XLink

XLink is able to link not only documents but also resources, which include documents, audio, video, database data, and any addressable information or services. While HTML links require editing the resource to make additional links, XLink does not require any write permission for the source. XLink can simply set the URI with the starting and ending point for the linking. XLink also provides multidirectional links (*extended* links) as well as the unidirectional links (*simple* links)—the traditional link on the Web. The links can be stored externally from the documents, they address by URI (*extended* link), or they can be inline. Traversal of "A" link in HTML usually replaces the document currently viewed. Traversal means "using or following a link for any purpose" [16]. The user may initiate traversal by clicking on the link, or the retrieving document may initiate it.

Figure 2.5 shows an example, which includes the features of XLink and XPointer. In the example, "<doc xmlns:xlink = "http://www.s3.org/1999/xlink">" associates the XLink namespace definition with the URI. The *extendedlink* element is a kind of extended link, which has full XLink functionality including arcs (*inbound* and *third-party*) and links with arbitrary resources. The other type of link is the *simple* link, which has only two participating resources.

In the *extendedlink* element, three sub-elements are embedded: they are two *locator* elements (XLink type) and an *arc* element (XLink type). The *locator* type element designates remote resources, whose location is denoted with the locator attribute, "href." The arc type elements represent the link traversal, which is usually a pair of start (*from*) and end (*to*) resources. The *locator* labeled "seaplace" uses an XPointer and the expression in the parentheses of this XPointer is an XPath expression. This XPointer points to the first *place* element of the second *sentence* in the first *animal* element of the *body* element. The "seareference" *locator* links to the first "places" element, which has the attribute *id* named "sea."

```
<?xml version="1.0"?>
<doc xmlns:xlink="http://www.w3.org/1999/xlink">

<head>
<title> Animals </title>
<extendedlink xlink:type="extended">
  <loc xlink:type="locator" xlink:label="seaplace"
       xlink:href="#xpointer(//body/animal[1]/sentence[2]/place[1])"/>
  <loc xlink:type="locator" xlink:label="seareference" xlink:href="#sea"/>
  <arc xlink:type="arc" xlink:from="seaplace" xlink:to="seareference"
       xlink:show="new" xlink:actuate="onRequest"/>
</extendedlink>
</head>

<body>
<animal name="whale">
  <sentence>Whales are mammals.</sentence>
  <sentence>Whales live in the <place>sea</place>.</sentence>
</animal>
<animal name="horse">
  <sentence>Horses are mammals.</sentence>
  <sentence>Horses live in the <place>land</place>.</sentence>
</animal>
</body>

<tail>
<reference>
<places id="sea">
  Sea is the continuous body of salt water covering the earth.
</places>
<places id="land">
  Land is the part of the earth not covered by water.
</places>
</reference>
</tail>

</doc>
```

Figure 2.5. An example XML document with XLink expressions

The *arc* element in the example has two remote resources for the traversal and is called a *third-party* arc. If the arc is from a local resource to a remote resource, it is the *outbound* arc. An *inbound* arc traverses from a remote resource to a local resource. The traversal attributes, "from" and "to," are the start and the end points of the link. The *show* and *actuate* attributes represent the behavior of the link. They designate the behavior of the ending resource of the arc. In the figure, the "new" value for the *show* attribute will open a new window when the traversal event has been requested. The *actuate* attribute sets "onRequest," and it constrains the traversal event. If the value of the *actuate* attribute was changed to "onLoad," the new window would be shown immediately on loading the starting resource.

CHAPTER 3

XML AND DATABASES

An XML document is a text document used for information exchange between application programs, typically through the Web. XML does not force any internal structure on the computer. However, from the viewpoint of efficiency of data manipulation, it is important to consider how to store and process the XML document. XML logically has a tree structure with elements and attributes, and is used both in the role of a data transport format and as a document markup language. Bourret [28] classifies the XML documents as either *data-centric* documents or *document-centric* documents. Data-centric documents are highly structured and have relatively small sized text elements. Document-centric documents have free-format text with some words marked-up. The performance of the XML document processing depends on the kind of XML document presentation: data-centric and document-centric documents have pros and cons relative to the different types of database.

Many studies were made to leverage the current database technologies to represent XML documents. Mapping XML to relational databases was one of the first methods used to store XML documents in existing databases. Many XML repositories have proposed ways to map XML to relational databases, because the relational database dominates the current database market and many applications are already developed on relational databases. Not only legacy applications, but also data-centric XML documents obtain benefits from using a relational database. Object-relational storage is a natural fit for XML storage, because its logical structure is similar to that of XML documents. Another approach for the XML storage is *semistructured* data. Semistructured data had been developed before the XML standard emerged. The XML format unsurprisingly has been applied to semistructured databases, because XML is a semistructured data format. *Lore* [21] is a well-known example of a semistructured data system to present XML documents as semistructured data.

3.1 Relational Database

3.1.1 Oracle Database System

Currently relational databases dominate the database market. Since XML emerged as a new standard for information exchange, many relational databases have tried to combine their databases and XML technologies. One of the initial answers from commercial databases is the XML-enabled Oracle 8i [29] from Oracle, which has the biggest market share in the current database business.

Mapping an XML document to a table or several tables is a primitive way of storing XML documents in a relational database. The elements, attributes, and names are mapped to the columns of tables, in ways depending on the functionalities of data usage and mapping design. This method is useful in transferring data between relational databases, but it is not applicable to sophisticated XML formats.

Oracle [30] provides a natural way of using object-relational support. The elements with attributes define *object* types that encapsulate data. Sets of object types and references to object types can form a model of classes. A class maps into a table. In Figure 3.3, *CAR_TYPE* is defined as an object and a single object maps to a table in this example.

Another approach for XML document storage is as a *Large Object* (LOB). A LOB column holds an XML instance. This type of storage is useful for document-centric documents. The XML instances in LOB columns can be indexed as in other texts. The new version of the Oracle database, *Oracle 9i*, provides *XMLType*—an object data type on a *character large object* (CLOB) column storage. XMLType supports XPath in SQL queries to extract elements and attributes of XML instances as follows:

```
SELECT c.car.extract('/car/model/text()').getStringVal() FROM cars c;
```

To improve performance, a key element for the XML document can be stored in another column and indexed. Without separate column indexing, extracting query performance is not acceptable in the case of a large number of rows. Updating elements or attributes in LOB is not possible in Oracle 9i, but Oracle 9i release 2 or later versions [31, 32] support updating part of the XMLType DOM located by XPath. There are additional packages and

```
create table CARS
(
ID    VARCHAR2(7),
STATE   VARCHAR2(2),
COMPANY  VARCHAR2(16),
MODEL   VARCHAR2(16),
TYPE    VARCHAR2(3),
YEAR    NUMBER(4),
COLOR   VARCHAR2(16)
)
/

create or replace view NEWCARS as
select SYS.XMLTYPE.CREATEXML('<CAR/>') "CAR"
from CARS;
```

Figure 3.1. An example for the XML mapping table and view to Oracle database

functions integrated in SQL which can be used in queries to wrap the data in columns and produce XML documents.

Other new major features in Oracle 9i release 2 is XML schema support and context based indexing on XMLType column. XML schema can be registered, and we can create and validate XMLType tables based on the registered schema.

Figure 3.1, Figure 3.2, and Figure 3.3 show an example that maps between an XML document and a relational table using Oracle 9i. The code in the first two figures represents the mapping from an XML document to a relational table. Figure 3.3 shows how to get an XML instance from a relational table. The *CARS* table in Figure 3.1 stores the elements and the attributes of the XML example from Section 2.1. The *NEWCARS* view shows XML instances from a relational table, *CARS*. When an XML document is inserted into a relational table, each element and attribute needs to be shredded. The trigger in Figure 3.2 substitutes the insert query to map the XML document into proper types of columns. From the relational table, an XML instance can be produced. In Figure 3.3 the *CARDOCUMENT* view generates an XML instance from a table using the *SYS_XMLGEN* function in the SQL query.

```
create or replace trigger CAREXPLOSION
instead of insert on NEWCARS for each row
declare
  CARID       VARCHAR2(7);
  STATE       VARCHAR(2);
  COMPANY     VARCHAR2(16);
  MODEL       VARCHAR2(16);
  CARTYPE     VARCHAR2(3);
  CARYEAR     NUMBER(4);
  COLOR       VARCHAR2(16);
  DOCUMENT    sys.XMLTYPE;
  ELEMENT     sys.XMLTYPE;
  NOT_A_CAR   exception;
  I           binary_integer;
begin
  DOCUMENT := :new.CAR;
  if (DOCUMENT.existsNode('/car') = 0) then
    raise NOT_A_CAR;
  end if;
  CARID := DOCUMENT.extract('/car/@id').getStringVal();
  STATE := DOCUMENT.extract('/car/@state').getStringVal();
  ELEMENT := DOCUMENT.extract('/car/company/text()');
  if (ELEMENT is not null) then
    COMPANY := ELEMENT.getStringVal();
  end if;
  ......
  ELEMENT := DOCUMENT.extract('/car/year/text()');
  if (ELEMENT is not null) then
    CARYEAR := ELEMENT.getNumberVal();
  end if;
  ......
  insert into CARS values (CARID, STATE, COMPANY, MODEL, CARTYPE,
CARYEAR, COLOR);
  exception
    when NOT_A_CAR then
      raise_application_error(-20000, 'Only car documents can be
stored in this column.');
end CAREXPLOSION;
/
```

Figure 3.2. A trigger example for the XML document insertion

```
create or replace type CAR_TYPE as object
(
"@id"     VARCHAR2(7),
"@state"  VARCHAR2(2),
"company" VARCHAR2(16),
"model"   VARCHAR2(16),
"type"    VARCHAR2(3),
"year"    NUMBER(4),
"color"   VARCHAR2(16)
)
/

create or replace view CARDOCUMENTS as
select
sys_xmlgen(
CAR_TYPE(
  C.ID,
  C.STATE,
  C.COMPANY,
  C.MODEL,
  C.TYPE,
  C.YEAR,
  C.COLOR
),
sys.xmlgenformattype.createformat('car')
) car
from CARS C;

SQL> select C.car.getClobVal() "car" from CARDOCUMENTS C;
car
--------------------------------------------------------
<?xml version="1.0"?>
<car id="J544XD" state="NY">
 <company>Toyota </company>
 <model>Corolla </model>
 <type>DX </type>
 <year>1996</year>
 <color>white </color>
</car>
```

Figure 3.3. An example for the XML extraction from the relational table

```
...
<dad>
  <dtdid>dxx_install/dtd/getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcolumn>
    <table name="car_table"> </table>
      <column name="year" type="decimal(4,0)"
       path="/car/year"/>
    </table>
  </Xcolumn>
</dad>
```

Figure 3.4. An example of DAD side table definition

3.1.2 IBM DB2 Database System

IBM's *DB2* database system [33] also supports XML documents for storage and query. To store XML documents in *IBM DB2 XML Extender*, an XML repository, *XML column* and *XML collection* options are available.

In the XML column option, the XML document is saved as *XMLCLOB*, *XMLVARCHAR* or *XMLFILE* type without shredding. Some elements and attributes can be stored in *side tables* and indexed for performance improvement. The side tables and index must be in a *Data Access Definition* (DAD). An example DAD for side tables is given in Figure 3.4.

In the XML collection option, an XML document maps into a set of relational tables and the mapping mechanisms between DTD and tables are described in a DAD. This option is useful for frequent updates to part of the data, extraction of only some part of the data, and relating to other relational data.

To publish XML documents from database tables, SQL queries with macros of a script language and stored procedures are used.

3.1.3 Microsoft SQL Server

Microsoft SQL Server [34, 35] is another relational database system that generates and stores XML documents through relational data. XML enabling features are executed in

middle tier applications, for example, *templates* and *XML views*. Templates are XML documents that include SQL queries executed against the database. XPath is supported in templates. An XML view of the relational data can be created by an annotated schema using an *XML Data Reduced* (XDR) schema. These annotations are used to specify a mapping between XML and relational tables.

To publish an XML document from a relational database, an SQL Server includes SQL extensions to produce query results as XML documents. There are three different ways to serialize SQL query results into XML: *RAW*, *AUTO*, and *EXPLICIT* modes. In RAW mode, each row of the query result maps into a name of an XML element, *row*, and each non-NULL column of the query result maps to an XML attribute. The example query may produce results as follows:

```
SELECT CarID, CarState FROM Cars FOR XML raw
```

```
<row CarID="J544XD" CarState="NY"/>
```

To produce query results with nested XML elements, SQL server provides AUTO mode. Each row maps to an XML element and each table alias is used for the element name. The order of the table names in the SELECT clause determines the nesting according to their appearance from left to right. The columns of the query results map to the XML attributes as in RAW mode. The example of an auto mode query may return the result as follow:

```
SELECT Dealers.DealerID, Dealers.DealerName, Customers.CustomerName
FROM Dealers, Customers
WHERE Dealers.DealerID = Customer.DealerID
FOR XML auto
```

```
<Dealers DealerID="ROMANOTOYOTA" DealerName="Peter Dan">
  <Customers CustomerName="Jungkee Kim"/>
  <Customers CustomerName="Bryan Carpenter"/>
  ...
</Dealers>
```

The EXPLICIT mode can produce any XML document from the relational database tables. The EXPLICIT mode query, which specifies the structure of the XML tree, produces a *universal table* which consists of a *tag* and a *parent*, column names, and a row ordering. For, example the query may produce the universal table and an XML instance as follows:

```
SELECT 1 as Tag, NULL as Parent, Dealers.DealerID AS
[Dealers!1!DelaerID],
  NULL as [Customers!2!name!element]
FROM Dealers
UNION ALL

SELECT 2, 1, Dealers.DealerID, Customers.CustomerName
FROM Dealers INNER JOIN Customers ON Dealers.DealerID =
Customers.DealerID
ORDER BY [Dealers!1!DealerID]
For XML explicit

Tag Parent Dealers!1!DealerID Customers!2!name!element
1   0      ROMANOTOYOTA       NULL
2   1      ROMANOTOYOTA       Jungkee Kim
1   0      SYRACUSEFORD       NULL
2   1      SYRACUSEFORD       Bryan Carpenter

<Dealers DealerID="ROMANOTOYOTA">
<Customers><name>Jungkee Kim</name></Customers>
</Dealers>
<Dealers DealerID="SYRACUSEFORD">
  <Customers><name>Bryan Carpenter</name></Customers>
</Dealers>
```

To produce a convenient relational view from an XML document, the *OpenXML rowset provider* is provided, similar to the *extract* function of Oracle 9i.

3.1.4 STORED

STORED (Semistructured TO RElational Data) [36] is one of several initial proposals for storing and querying XML documents mapped to relational database systems. STORED utilizes a combination of relational and semistructured techniques to manage XML documents. It tries to discover the most frequently occurring sub-trees from XML documents and maps the extracted sub-trees to relational tables. The remaining part of XML documents is stored in a semistructured *overflow* graph. For example, the XML document and the graph in Figure 3.5 will produce a relational table *car* with attributes, a plate and a driver. The second driver, Mary, might be stored in the overflow repository.

A declarative query language is defined and this language expresses the structure in both the input and the output of a query. Because the language is non-recursive, the elements cannot have an arbitrary number of sub-elements in the same format. If a DTD is available, the overflow queries are expected to be executed faster than they would be without a DTD.

When STORED generates a mapping, it requires various parameters: the maximum number of tables, the maximum number of attributes per table, the maximum disk space, the *collection size threshold*, and *minimum support*. The collection size threshold designates the boundary between *small sets* and *collections*. The minimum support parameter is required for the mining algorithm. The mining algorithm determines which elements of the XML document should be stored in relational tables and which elements should be saved in overflow. The mining processes measure which path prefixes and bodies are frequently occurring in XML documents and queries. From those numbers, the best relational mapping will be selected taking into account the number of matching values and the maximum disk space.

Since the queries and update are performed against the original XML documents rather than mapped tables or overflow, STORED should *rewrite* them into queries and updates over relational tables and overflow storage. The rewrite algorithm of the STORED system, *inversion rules*, converts the queries over XML documents to take table columns and add tree structure to the data layout.

3.1.5 Relational Mapping Technology from University of Wisconsin

Shanmugasundaram and others [37] suggested a different approach for mapping XML documents to relational database systems. They use a DTD to generate a relational schema. There is no consideration of the query workload. XML documents are parsed, matched to DTDs, and loaded to relational database tables. They used an IBM DB2 database for the relational tables. For querying the data, semistructured queries are translated to SQL queries and the results are converted to XML.

When generating relational schemas from DTDs, DTDs are simplified with a set of transformations that is more restricted than that of STORED. Transformations are performed for flattening, simplification, and grouping as in the following examples:

```
Flattening: (e1, e2)* -> e1*, e2*
Simplification: e1** -> e1*
Grouping: ..., a*, ..., a*, ... -> a*, ...
```

Because of the simplification, the order of elements of the original XML document may be lost though additional fields for some elements were suggested for keeping the order. A DTD can be expressed as a graph. The simplified DTD graphs can be converted to relational schemas using one of three proposed methods: the *Basic Inlining* technique, the *Shared Inlining* technique, or the *Hybrids Inlining* technique.

The Basic Inlining technique creates separated relations for each element of an XML document, because all elements of a DTD can be a root of an XML document. Each relation has an ID that acts as the key field. All sub-elements and attributes of the element in the relation are inlined, but there are two exceptions. Multiply occurring subelements—the nodes below "*"—and recursive referenced elements are not inlined. They have separate relations. The recursive element will also have the parent reference. The XML fragment in Figure 3.5 may produce the relations as following:

```
cars(carsID: integer)
cars.car(cars.car.carID: integer, cars.car.parentID: integer,
cars.car.plate: string)
cars.car.plate(cars.car.plate.plateID: integer,
```

```
cars.car.plate.parentID: integer, cars.car.plate: string)
cars.car.driver(cars.car.driver.driverID: integer,
cars.car.driver.parentID:integer, cars.car.driver: string)
```

The Shared Inlining technique expresses each element as a relation and avoids the redundancy of the Basic Inlining technique. The multiple relations for elements with several parents in Basic Inlining are stored in each relation in Shared Inlining. A new relation for those elements is created and shared. All the nodes with in-degree of one will be inlined into columns of the tables for their parent elements. The Shared Inlining for Figure 3.5 may produce the following relations:

```
cars(carsID: integer, cars.isRoot: boolean)
car(cars.car.carID: integer, cars.car.parentID: integer,
cars.car.parentCODE: integer, cars.car.plate: string, cars.car.plate:
string)
driver(driverID: integer, parentID:integer, cars.car.driver: string)
```

However, Shared Inlining can require more join operations on some particular elements comparing to Basic Inlining.

The Hybrid Inlining technique is similar to the Shared Inlining technique, but it has additional inlining which is not included in Shared Inlining. The extra inlining elements in this technique are not for recursive or cyclic links.

The performance for the Basic Inlining technique is very poor. There are trade-offs between using the Shared Inlining technique and the Hybrid Inlining technique.

3.2 Semistructured Database

Research on semistructured data initially looks for an efficient way to describe data with no fixed schema, for example, the information on the World Wide Web. The relational or object-oriented database systems have schemas, and all the instances in the systems should be created under the rules of the schema. Semistructured data have no explicit explanation of the structure, but they include direct descriptions of the data with a simple syntax. The *Object Exchange Model* (OEM) is a typical semistructured data model. It was designed for

exchanging data between heterogeneous systems and originates from the Tsimmis [38] data integration project. An OEM object has four components: *label*, *oid*, *type*, and *value*. Label is a character string and oid is the identifier of the object. Type can be an atomic type or a complex type. If the object type is complex, the value is a set of oids. The value of an atomic type is one of the base types—*integer*, *string*, *image*, *sound*, etc. Therefore, OEM data form a graph in which the nodes are the objects and the labels are generally attached to edges, though the initial system was labeled on nodes.

3.2.1 Lore

Abiteboul and others [39] described three approaches to develop a database management system for semistructured data: "building an application on top of existing relational or object database systems," "using a low-level object server," and "building the system from scratch." The *Lore* (Lightweight Object Repository) system from Stanford University [21] uses the last approach and is a typical database system for semistructured data. The Lore system consists of the API, query processors, and data managers. A graphical user interfaces provides queries and views of data by the users. *Lorel* is the query language of Lore. The queries presented with Lorel are parsed, preprocessed, transformed, and optimized in query processors. There are several managers and tools for data handling. The OEM objects can be saved to or fetched from files though an object manager. The XML documents are mapped and kept in OEM forms internally. To describe the structures of the stored semistructured data, DataGuides [40] were introduced in Lore.

A DataGuide is a *concise* and *accurate* summary of the structure of a specified data graph. It describes every unique path of the original data graph only once. Every label path appearing in the DataGuide exists in the original graph to reserve the accuracy. The logical structure of a DataGuide is a directed acyclic graph. A *target set* is the set of all objects that a given label path in a graph reaches. A node in the input graph may appear more than once in the target set, because the node can be reached from several different edges. The target sets containing IDs are the *annotations* of the label paths that designate the nodes of the DataGuide. Goldman and Widom [40] assert that generating DataGuides over the input graphs is similar to the conversion from a non-deterministic finite automation (NFA) to a deterministic finite automation (DFA). They expect that the conversion would be finished

within a reasonable time and they did not see any exponential time or space problem during their experiments. However, those annotations mean that multiple label paths may exist to reach the same object. In Figure 3.6, a data graph and two DataGuides of the graph are shown.

We can reach the node 2 with path A from the data graph and DataGuide 1, but the same node attains through both A and B paths in DataGuide 2. To avoid the confusion as in DataGuide 2, a class of DataGuides is defined and named *Strong DataGuide*. The main aspect of strong DataGuides is that "each set of label paths that share the same (singleton) target set in the DataGuides is the set of label paths that shares the same target set" in the data graph.

3.2.2 Other Native XML Databases

The terminology *native XML database* is used commercially for semistructured databases. It was introduced by Tamino [41] as a marketing slogan [28]. Some other companies followed this terminology for their products and the "native" term has become more popular than "semistructured" in recent years. There are many native XML database products including open source projects like Apache Xindice [42]—the successor of dbXML [43]—and Berkeley DB XML [44] from Sleepycat—an extension of Berkeley DB [45]. In Xindice the data is stored in binary format using a "proprietary storage" [28]—indexed or compressed file. In Berkeley DB XML, XML documents are stored in collections—comparable to tables in a relational database—and the records in storage are "key/value" pairs in Berkeley DB. The update of a fragment of an XML document is available. XPath 1.0 is implemented in the query language. Many other XML database products are well classified in [46]. Few details are known about the storage mechanism of many native XML databases.

```
<cars>
  <car>
    <plate>J544XD</plate>
    <driver>Jake</driver>
  </car>
  <car>
    <plate>S7RDG</plate>
    <driver>Tom</driver>
    <driver>Mary</driver>
  </car>
  <car>
    <plate>53G123</plate>
    <driver>Bryan</driver>
  </car>
</cars>
```

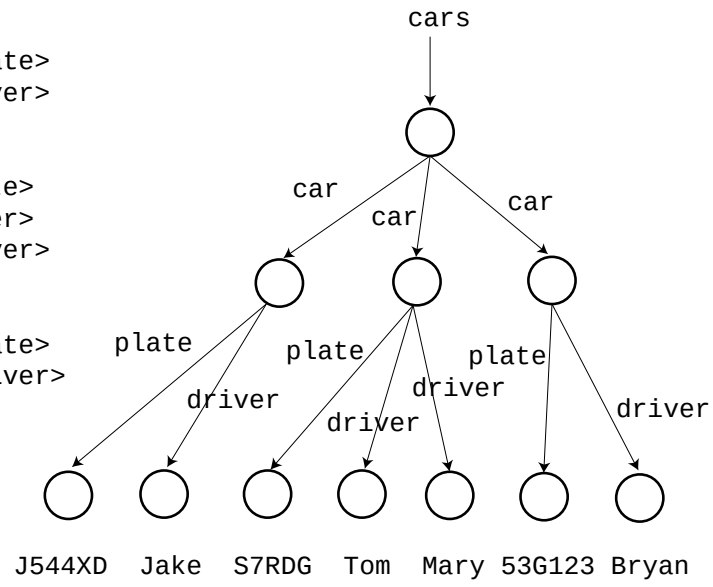

Figure 3.5. An example of an XML document and a graph

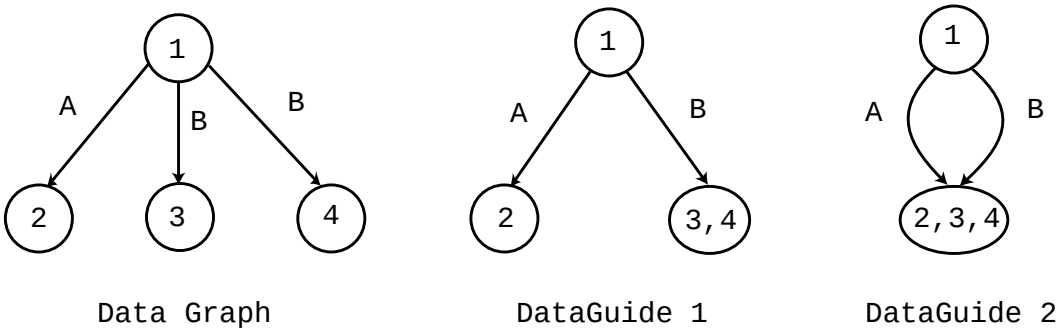

Figure 3.6. A data graph and two DataGuides

CHAPTER 4

P2P AND MESSAGE-ORIENTED COMMUNICATION

4.1 P2P

Peer-to-peer (P2P) technology is emerging as a new model to organize distributed systems for file sharing, distributed computing, system integration, or information search. However, P2P is not a new idea for network communication. In an earlier age of the Internet, many network systems depended on P2P technology. Usenet and the Domain Name System (DNS) were early examples of P2P based systems. However, recent Usenet departs from pure P2P by providing only selective Newsgroups and messages, because the Internet Service Provider (ISP) does not serve all Newsgroups, due to the huge volume of News messages. The newer *Freenet* [47] restores the original P2P architecture. It is designed to protect itself from censorship using encrypted files and "route-through" [48] data communication. It serves a full list and the server cannot block a particular Newsgroup. DNS is a mixed form of P2P and a hierarchical model. Searching a particular name is referred from lower level to higher level name servers. The name server has the role of server as well as client.

4.1.1 Napster

Napster [8] was the most famous P2P file sharing system prior to the lawsuit that closed it down. In Napster, the metadata of the shared files are indexed on a centralized server. All the requests for locating files are served through this server. Only the actual file transfer occurs between peers. When a user joins the system, the user connects to the server and sends a list of possessed sharable files. To obtain target files from Napster, a "wanted" list is sent to the server. The result of the query returns the IP addresses of peers possessing

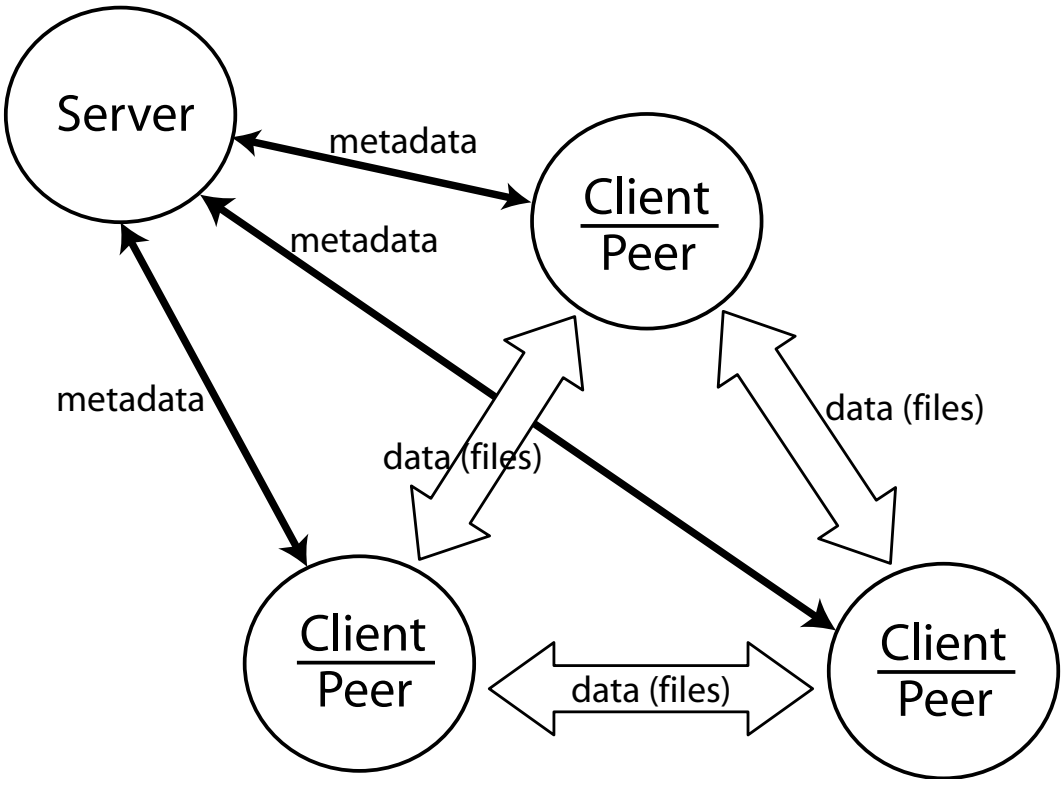

Figure 4.1. Napster Architecture

the wanted files. The user can select one of the returned IP addresses and download the file. When a user disconnects from the server, the list of sharing files possessed by that user is deleted from the server [49].

4.1.2 Gnutella

Gnutella [50] is a pure P2P network protocol [51]. The initial list of hosts is usually obtained from a transitory Web lookup. A *servant* (node or peer) broadcasts a *Ping* message to find the active hosts forming the initial network. Those hosts are neighbors and the actual query is broadcast to the neighbor hosts. The neighbor servants send the query to their own neighbors again, even when they have the target file. All the messages are forwarded to the neighbors within the limited number of hops defined by the Time to Live (TTL) in the message header. If the desired files were found, the servant sends back the matched result set

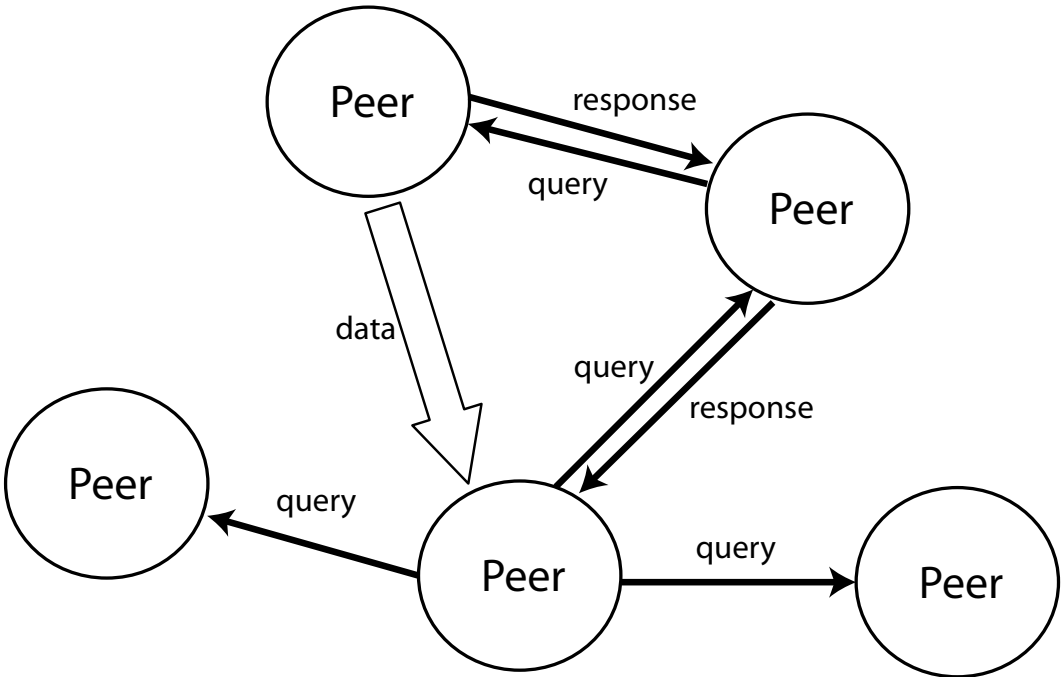

Figure 4.2. Gnutella Architecture

to the neighbor through which it received the query. The request servant selects the file from the result set and downloads the target file directly from the possessing host in the same way as in HTTP download. Due to the Breadth-First-Search (BFS) mechanism of query in the Gnutella protocol, Gnutella may cause network traffic to significantly increased. A number of studies have been focused on reducing such inefficient network usage. Distributed Hash Tables (DHTs) [52, 53, 54, 55] is one approach that reduces network communication. In DHTs, a key is mapped onto a node, which is assigned through a hash table.

4.1.3 Supernode

The *Supernode* (*Hub, Ultrapeer, Ultranode*, or *Reflector*) architecture is introduced to the unstructured peer-to-peer network for better performance. *Fasttrack* [56] organizes a *hybrid*—centralized and decentralized—form of network topology as in Figure 4.3. Instead of centralized control, a peer node that has enough bandwidth and storage is voluntarily selected as a supernode. The supernode has a larger number of connections than ordinary

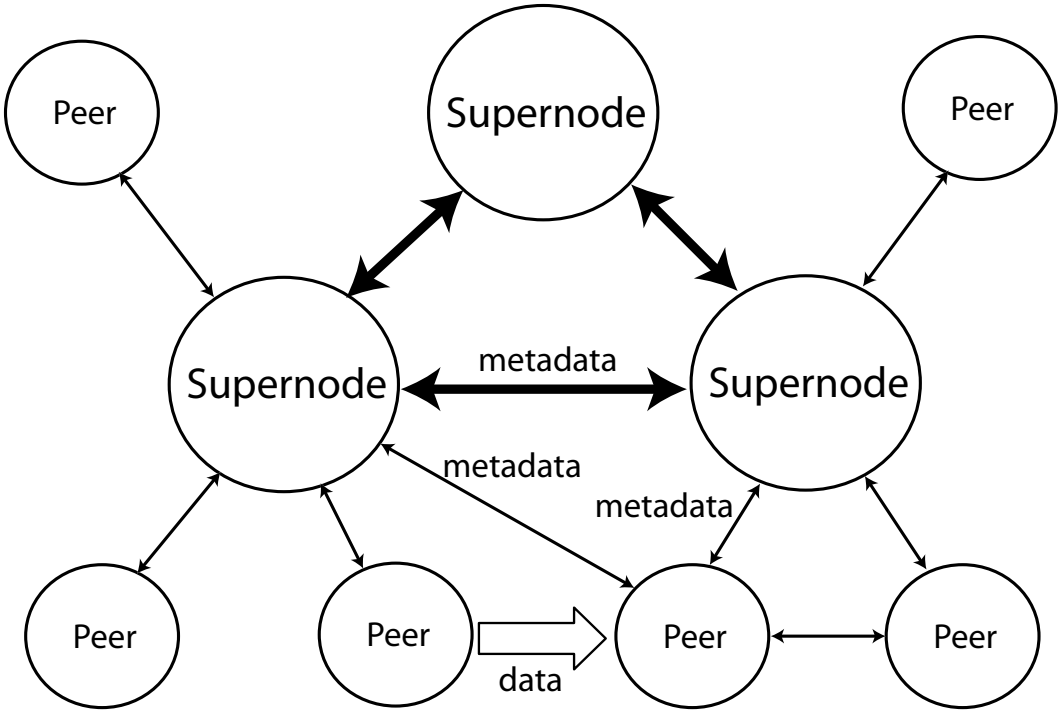

Figure 4.3. Fasttrack Architecture

nodes. Most peers act the same way as in Gnutella and also query the supernode to find target data. The large amount of cached metadata in supernodes dramatically reduces the network traffic. From the list of target data from peers including supernodes, the selected target files can be downloaded through the simplified HTTP protocol.

4.1.4 JXTA

JXTA is the first open, platform and language independent protocol for general P2P communication and collaboration between network devices [10]. JXTA makes it possible to organize a virtual network in which heterogeneous peers can share files, communicate with each other, and collaborate on top of the JXTA protocol.

A unique JXTA ID is allocated to each network resource. The JXTA ID is not related to the Internet Protocol (IP) address, and the ID is not changed even if IP addresses are dynamically allocated. Any network entity that executes the JXTA protocols [57] is called

a peer. Peers may be not only computers but also network devices. Each peer has network functionalities that are "independent and asynchronous" from other peers. A peer endpoint links a peer. Several peer endpoints, which represent various network connections, can reach the same peer. A group of peers that are interested in the same category can form a *peer group*. A peer group is identified with an ID for the group. Peers in the different domains, possibly separated by a firewall or Network Address Translation (NAT), can belong to the same group. A peer is allowed to have membership of multiple peer groups.

Messages are used to communicate between JXTA peers. An ordered sequence of named and typed elements forms a message. There are two formats for the message—XML and binary—but only a binary format is used for the physical transfer. The messages are transferred through *pipes*, which is the JXTA virtual abstraction for a network connection. There are three kinds of pipe connections—a *point-to-point* pipe, a *propagate* pipe, and a *secure unicast* pipe. A point-to-point pipe is a one-to-one connection. A propagate pipe provides one-to-many connections. A secure pipe is a point-to-point communication through a secure network channel. All the pipe operations are based on "asynchronous and unidirectional" communications. Bi-directional communications are provided on top of the pipe service. Multicasting may be used in the propagate pipe.

An *advertisement* is an XML-based metadata script, which describes JXTA network resources—peer, peer group, pipe, service, and other core resources. The JXTA *resolver* is a generic object discovery to resolve any kind of information used in typical distributed systems. All actions of resolution are integrated into the discovery of advertisements. The advertisement search mechanism depends on the policy of the application, though JXTA protocol provides a substitutable protocol framework for resolution. The resolver service includes query send and response, query propagation, and security functions—authentication and verification of credentials. Each advertisement has an expiration time, which is extensible.

The *Rendezvous* super-peer is the optional but default policy model for resolution in a JXTA network. The Rendezvous peer is the same as other peers but has an additional cache of advertisement indexes. Through the Shared Resource Distributed Index (SRDI) service, non-rendezvous (edge) peers can publish their advertisement indexes on the rendezvous peers. The physical location of a peer does not affect the qualification of a rendezvous

peer. When a peer queries an advertisement, it sends a request to its rendezvous peer. If the rendezvous peer does not have the index of the query, it propagates the query to the next rendezvous. However, from JXTA version 2.0 [58], the propagation is transferred only among the rendezvous peers.

4.2 Message-Oriented Communication

Message-oriented communication is a way of communicating between processes. *Messages*, which correspond to *events*, are the basic units of data delivered. Tanenbaum and Steen [59] classified message-oriented communication according to two factors—*synchronous* or *asynchronous* communication, and *transient* or *persistent* communication. In synchronous communication, the sender blocks waiting for the receiver to engage in the exchange. Asynchronous communication does not require both the sender and the receiver to execute simultaneously. So, the sender and recipient are *loosely-coupled*. The amount of time messages are stored determines whether the communication is transient or persistent. Transient communication stores the message only while both partners in the communication are executing. If the next router or receiver is not available, then the message is discarded. Persistent communication, on the other hand, stores the message until the recipient receives it.

A typical example of asynchronous persistent communication is Message-Oriented Middleware (MOM) [60]. Message-oriented middleware is also called a *message-queuing system*, a *message framework*, or just a *messaging system*. MOM can form an important middleware layer for enterprise applications on the Internet. In the *publish and subscribe* model, a client can register as a publisher or a subscriber of messages. Messages are delivered only to the relevant destinations and only once, with various communication methods including one-to-many or many-to-many communication. The data source and destination can be decoupled under such a model.

The Java Message Service (JMS) [61] from Sun Microsystems provides a common interface for Java applications to MOM implementations. Since JMS was integrated with the recent version of the Java 2 Enterprise Edition (J2EE) platform, Enterprise Java Beans (EJB)—the component architecture of J2EE—has a new type of bean, the message-driven

bean. The JMS integration simplifies the enterprise development, allowing a decoupling between components.

NaradaBrokering [62] is a MOM implementation that supports JMS and JXTA connections. It is also capable of building distributed networks of cooperating brokers. The publish and subscribe model is a basis for the communication. With the topic-based paradigm, a client can publish or subscribe on a specific topic and only receive the relevant messages. The system understands topics and utilizes them to deliver messages to the proper destinations. After topic subscription, a client can be disconnected and the system ensures delivery at the next connection. When a client connects again to the system after a prolonged time, it can connect to a new local broker rather than the remote broker that it connected to before. This will reduce network usage by avoiding the relay of the target messages from the original broker through several hops.

CHAPTER 5

HYBRID KEYWORD SEARCH

Our hybrid keyword search combines metadata search with a traditional keyword search over unstructured context data. Each chunk of unstructured data, usually represented by a file, has an assigned metadata. We use XML—the de facto standard format for information exchange between machines—as a metalanguage for the metadata. To demonstrate the practicality of the hybrid keyword search, we design and evaluate hybrid search systems based on a native XML database and a file system based text search library, as well as a market-leading relational database management system that integrates XML and text management.

5.1 Motivations

Our development of a hybrid search [63] is primarily motivated by the need to search a large legacy body of educational documents with available metadata [64]. While keyword-only searching is broadly used to search information, search engines often produce undesirably large search lists. Those engines use well-known information retrieval (IR) algorithms and technology [65, 66], and they usually have no schema that may narrow the search category.

Another paradigm for search engines is based on relational database management systems (DBMS). The relational DBMS provides an expressive and powerful query language to extract information, but querying requires knowledge of the schema. The complexity and difficulty of formulating a database query naturally motivates the emergence of keyword search in databases [4, 5]. While a keyword based search is simpler than, say a SQL query on the database, it loses the inherent meaning of the schema.

Our hybrid search paradigm is fundamentally based on keyword search for the individual document, and adds a supplemental search on the metadata attached to each document.

We expect this paradigm not only narrows the search category but also supplies semantic information over a legacy keyword-only search system. A collection of hybrid search services may provide low-cost scalability on a distributed environment, and this issue will be discussed in later chapters.

5.2 Unstructured Text Data and XML Metadata

Generally in a computer system, a chunk of unstructured data is a file containing this data. The format of the file can be pure text, *Microsoft Word*, *Adobe Portable Document Format* (PDF), *PostScript*, or any other format of binary or text document for which a program is able to extract text. For information retrieval, indexing of the text data is essential for a keyword search within an appropriate time. In a search service based on file systems, text filtering may be necessary to support the various data formats. Through the filter, we can obtain documents with a unified format—the pure text—and the document indexing program is uniformly applied to this text data. The market leading commercial database systems—IBM DB2, Microsoft SQL Server, and Oracle—have integrated text management in their systems, and their query languages include text search syntax [67, 68, 69].

The metadata to describe the information content of the data may be structured or semistructured. We use XML, a semistructured data format, as a description language for the metadata. This is because XML provides a unified format among heterogeneous databases. Similar to the case of unstructured data repositories, metadata storage can be based on low-level file systems. To extract information from XML instances in files, parsing and matching programs are needed. The extracted metadata information is combined with the query result of content search against the unstructured data. The relational database management systems are able to keep XML instances in their relational tables by the mapping paradigm. XML-enabled relational databases [33, 29, 30] are useful for the metadata storage, because mapping and querying features are embedded in their database management systems. Native XML databases have excellent features for manipulating XML instances, but they are document-centric.

The initial objective is to achieve both content and metadata search against existing documents associated with separate metadata. Existing searches focused on keyword-only

or XML-instance-only search. We address the hybrid search with two contrasting approaches. One of our approaches utilizes a relational database management system that supports XML and text management, and the combined query results are obtained from nested SQL subqueries. The other system depends on two open source programs, and hashing is used for joining search results. Detailed designs and evaluations are given in the following sections.

5.3 Hybrid Search in an XML-enabled Relational Database Management System

Relational database technology has matured over some decades of development and testing. It is characterized by a complex data model and effective query language. Since XML was introduced as a data format for exchanging information, many relational database management systems have integrated XML data format as reviewed in chapter 3. The first experimental implementation for our hybrid search utilizes the Oracle DBMS—a relational database management system.

5.3.1 Data Relationship and Nested Query

To resolve hybrid search in relational databases, we devise a simple relationship as in Figure 5.1. One entity set is allocated for the metadata described in XML and the other entity set is for the contents stored in external files. The primary key attribute for the metadata entity set is a naming and directory for the each file and the primary key is a file name for the contents entity set. From this relationship, we can obtain the hybrid search result using a nested subquery—for example:

```
SELECT f.filename
FROM FileContents f, Metadata m, Designates d
WHERE CONTAINS(f.TextContents, 'aKeyword', 1) > 0 AND
      f.filename = d.filename AND
      d.namingdirectory = m.namingdirectory AND
      m.namingdirectory IN
      (SELECT m.namingdirectory
       FROM Metadata m
```

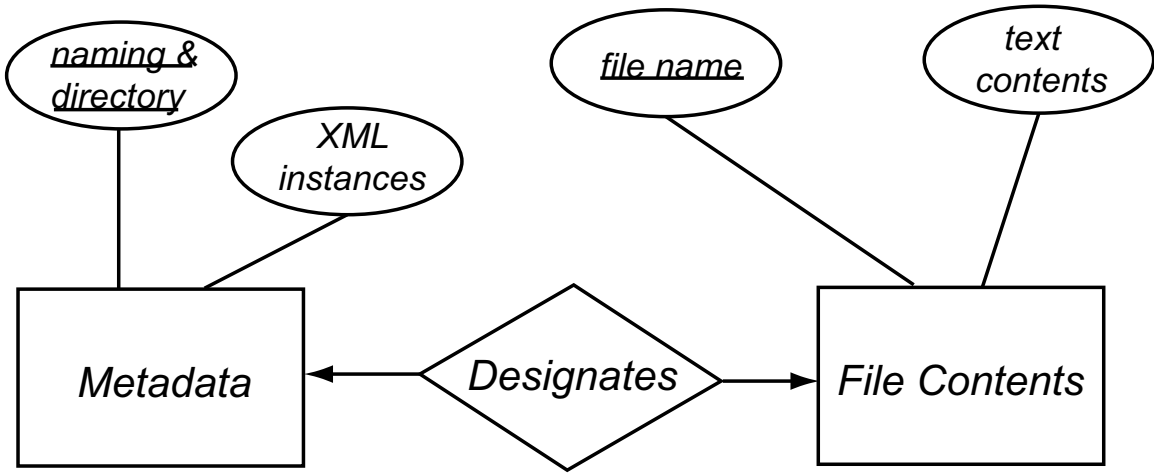

Figure 5.1. E-R diagram of hybrid search

```
WHERE m.xmlinstances.EXTRACT('anXPath').GETSTRINGVAL()
      LIKE '%partOfText%');
```

The nested subquery, which is a *SELECT-FROM-WHERE* expression within parentheses, collects a set of partial word matches of text elements in an XML instance. The *EXTRACT* function retrieves a specific node with a given XPath from an XML instance. We use the approximate match for the text node using the operator *LIKE* in this example. The *IN* connective tests for set membership of the matched naming and directory attribute in the Metadata entity set. The *CONTAINS* operator produces a number between 0 and 100 that specifies how relevant is the text row to a given keyword in the example. The number 0 means there is no match found in the row. Oracle utilizes an inverse frequency algorithm based on Salton's formula for word queries [70]. For example, 34 appearances of a query word in one document are necessary to score 100 in a document set.

5.3.2 Case Study: Hybrid Paper Search

The initial demonstration of the hybrid search is in a simplified model—a paper search. This is a test prototype for evaluation of the hybrid search. The paper search is a content

```
<bibliography>
<authors>
  <author>J. Kim</author>
  <author>O. Balsoy</author>
  <author>M. Pierce</author>
  <author>G. Fox</author>
</authors>
<title>Design of a Hybrid Search in the OKC</title>
<source>Proceedings of the International Conference on IKS</source>
<year>2002</year>
</bibliography>
```

Figure 5.2. An Example XML Instance

search across various types of documents. Each document has metadata represented as an XML instance. An example XML instance is shown in Figure 5.2.

Two relational database tables are used for metadata and documents. For the XML instances representing the metadata, the XMLType of Oracle 9i is used. A column with the *BFILE* large object type is used for the external document table. Those large object rows are indexed using Oracle Text—a text management system integrated into Oracle DBMS. Through an Oracle Text index, we can search the target content. The relationship is the same as in Figure 5.1, but the document table has another attribute for a document type—*BINARY* or *TEXT*. This attribute is necessary for the filtering option of Oracle Text. Oracle Text filters binary files to pure text instances before making an index. A one-to-one relationship set is used for the relation between the paper document and metadata, but a one-to-many or many-to-many relationship set could be used for other applications. The relationship table is not necessary in one-to-one relationship, but it is essential to decompose relations to avoid anomalies in one-to-many or many-to-many relationships. With two data tables and a relationship table we can query keywords in the content, which can be associated with particular metadata through nested subqueries. For example, we can find documents with a keyword "XML" and published in 2002. Figure 5.3 shows the relational schema of our hybrid paper search.

```
papers(paperND: string, descriptions:  XMLType)
paperfiles(filename:  string, doctype:  string, contents:  BFILE)
filelocator(paperND: string, filename:  string)
```

Figure 5.3. Relational Schema of Hybrid Paper Search

5.4 Hybrid Search in a Native XML database with a Text Search Library

Our second experimental implementation is not based on a relational DBMS. It uses two different open source projects. Apache Xindice [42] is a native database for XML instance repositories and XML query processing. The second Apache project, Jakarta Lucene [71] is a text search engine. This is used to manage context query over unstructured data in our hybrid search. The query processing architecture is shown in Figure 5.4.

To associate an XML instance with an unstructured document, we assign the file name for the document as the key of the XML instance. For example, an XML instance in a file named "pt1.xml" can associate with an unstructured data in "apaper.pdf" file when inserting an XML instance into the data collection as follows:

`xindice add_document -c /a_collection_path -f pt1.xml -n apaper.pdf`

The assigned key—the file name of unstructured document in the example—is returned as an attribute value on the root element of the XML created by executing an XPath query. An example of a root element of the query result is as follows:

```
<bibliography src:col="/a_collection_path" src:key="apaper.pdf"
xmlns:src="http://xml.apache.org/xindice/Query">
```

The key and result XML tuples are stored in a hash table. Another keyword search against unstructured documents returns names of documents which include the given keyword in the text. The returned names are mapped in the hash table and the combined results are collected. We use an unsynchronized hash table of class '`java.util.HashMap`' in our Java program.

For efficient text search, Jakarta Lucene provides an index class along with a stop word filtering analyzer class. The analyzer filters out stop words—articles and other words that are meaningless to the search. Some binary format files should be converted to pure text files

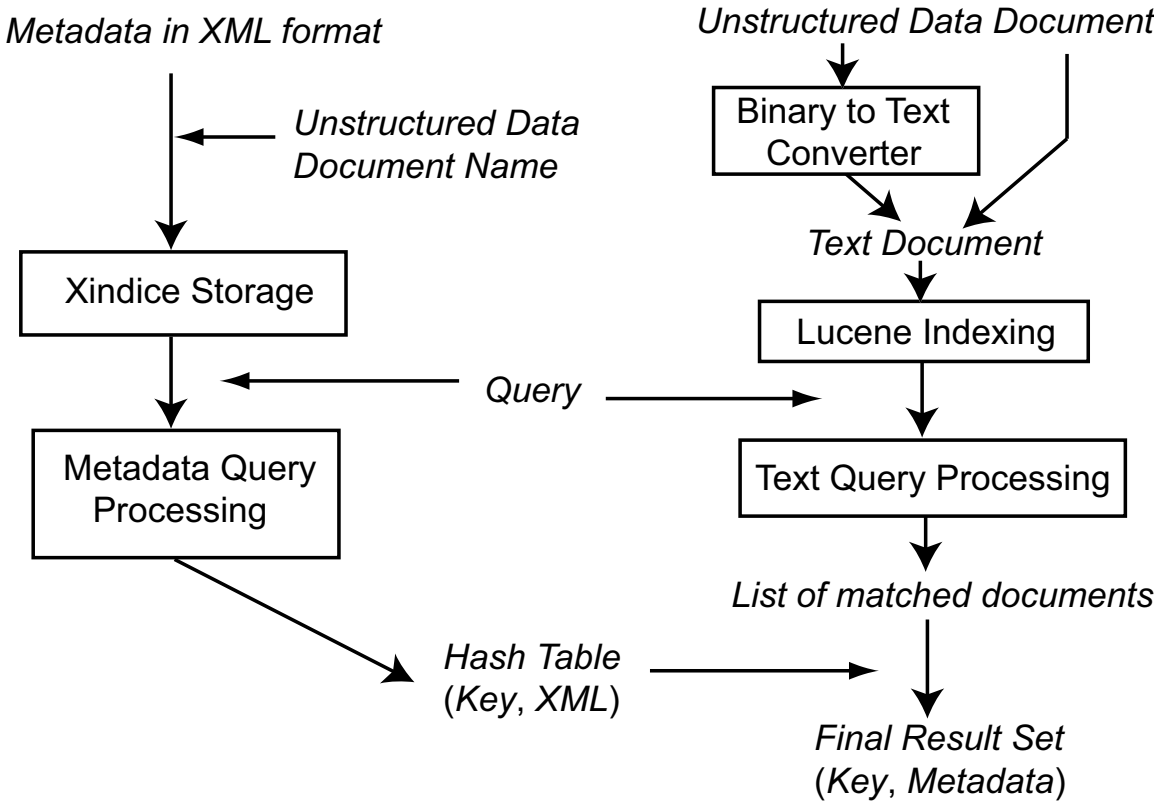

Figure 5.4. Query Processing Architecture

before indexing. We pass the binary file name as well as the text file name as parameters to the index generating class in order to indicate the original document format.

5.5 The User Interface and Result Transformation

Figure 5.5 shows the browser-based user interface of our hybrid search for both DBMS-based and non-DBMS based systems. One or more metadata fields are passed to the search program as metadata query parameters, along with keyword query parameters for text search. Query parameter passing in a non-DBMS system is shown in the middle of Figure 5.4 and those parameters divide into metadata and unstructured data query processing.

The metadata in our system is represented in XML, and it is necessary to convert this to a more user friendly format. We use the Extensible Stylesheet Language for Transformations

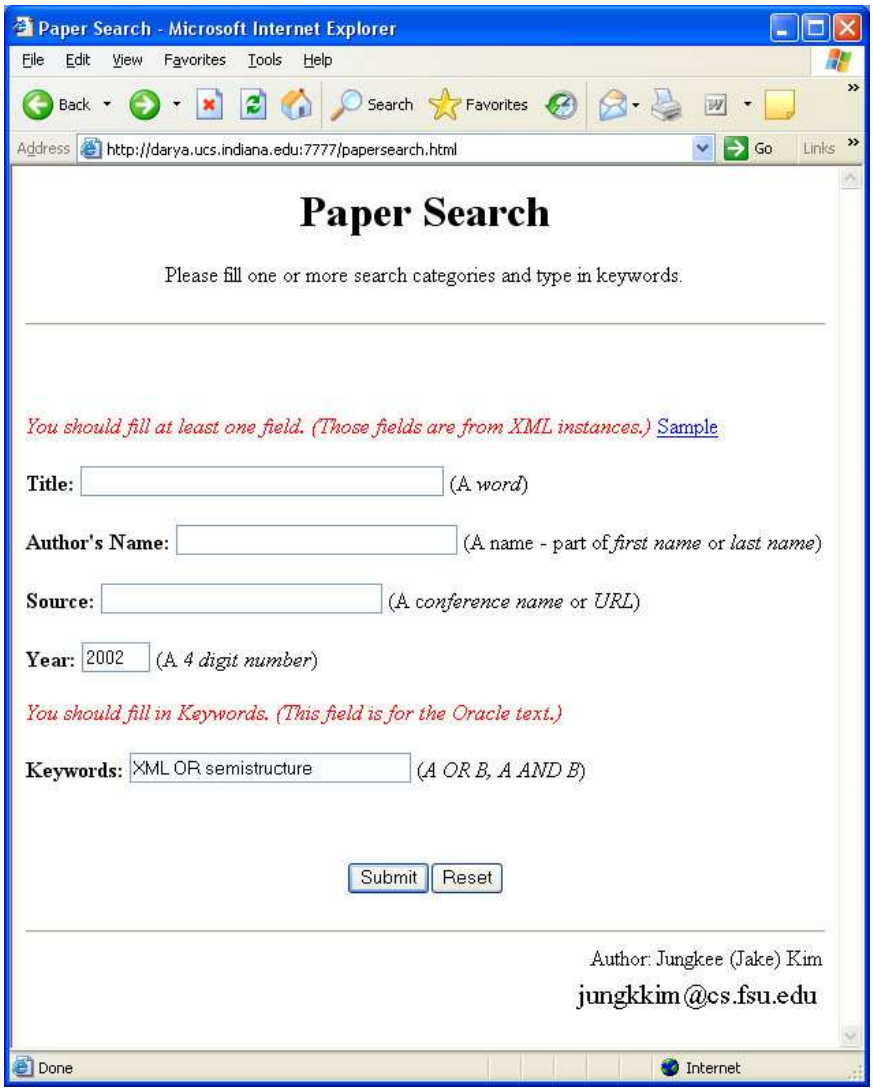

Figure 5.5. Browser-based User Interface for Hybrid Paper Search

(XSLT) [27] to convert XML instances to HTML or text instances. The XML instances of our hybrid paper search are converted to HTML instances through an XSLT stylesheet like the one in Figure 5.6. Figure 5.7 shows an example of query result screen.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method = "html"/>
    <xsl:template match="bibliography">

<B>Author:</B> <xsl:for-each select="authors/author">
    <xsl:value-of select="."/>  
</xsl:for-each> <BR/>
<B>Title:</B>  <xsl:value-of select="title"/> <BR/>
<B>Source:</B> <xsl:value-of select="source"/> <BR/>
<B>Year:</B> <xsl:value-of select="year"/>

    </xsl:template>
</xsl:stylesheet>
```

Figure 5.6. An XSLT Example of Hybrid Paper Search

5.6 Experimental Performance Comparison

To establish that our approach is qualitatively practical, we evaluate the performance of the hybrid keyword search. In this section, we will measure the experimental performance of the metadata query processing and the combined query processing for both XML-enabled relational DBMS and native XML databases. This measurement will show how our architecture works at a local level. The later chapters will introduce our possible solutions to the limitations of a local implementation.

5.6.1 Metadata-only Query Performance

We first evaluate the performance of the XML query processing in an XML-enabled relational DBMS—Oracle 9i. For Oracle we report the basic performance, and performance using the different kinds of indexing. The function-based indexing is based on evaluation of the specific function for each row data. Oracle Text indexing depends on the context search over XML. For example, the following SQL statement creates a function-based index on the text value of the *Reference* element.

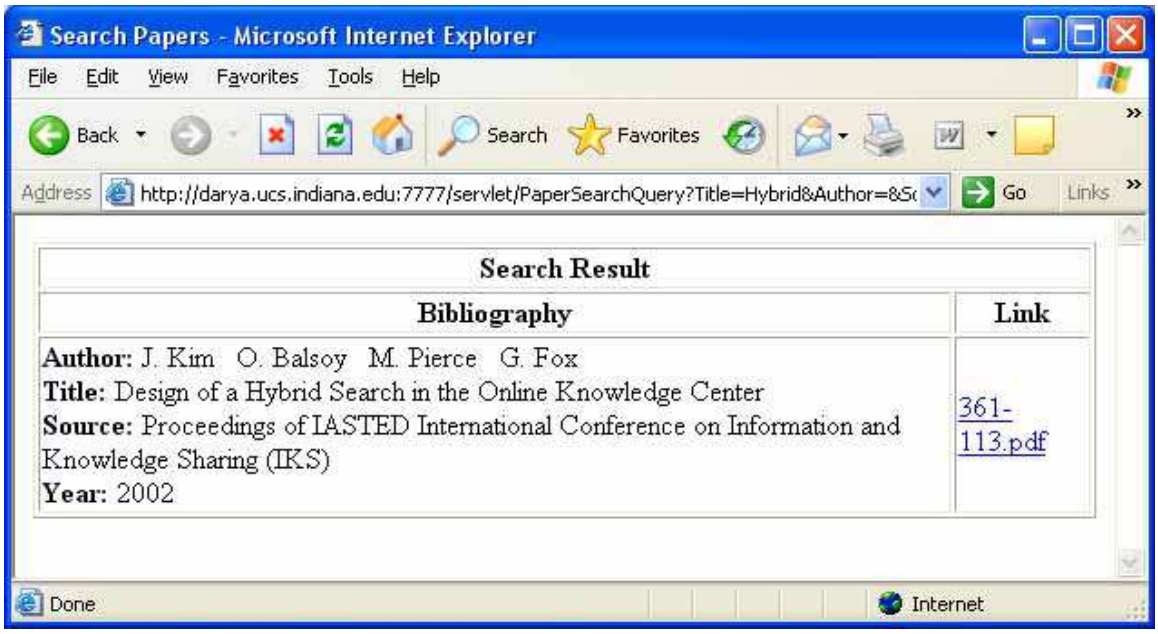

Figure 5.7. An Example of a Query Result Screen

```
CREATE INDEX po_index ON purchaseorder
       (podocument.extract('//Reference/text()').getStringVal());
```

Oracle Text indexing on the other hand could be created by the following SQL statement:

```
CREATE INDEX po_ref_index ON
       purchaseorder(PoDocument) indextype is ctxsys.context;
```

We will see that function based indexing is not very useful in our examples.

We initially use 10,000 data-centric XML instances presenting the purchase order available in an Oracle sample code [72] on a 1,700 MHz Pentium 4 PC with 512 MB of memory, running Microsoft Windows 2000. Oracle 9i has improved the indexing over XML repositories in Release 2. We compare the performance of the XPath query against the metadata—XML instances in our test—for those two versions in Table 5.1. A further experiment is made using another data set on a different machine.

We use 100,000 XML instances extracted from *DataBase systems and Logic Programming* (DBLP) XML record [73] on an Athlon 1800 machine with 512 MB of memory, running a

Table 5.1. Time for querying an XPath

Time	Oracle 9.0	Oracle 9.2	Oracle 9.2 with function-based index	Oracle 9.2 with Oracle Text index
Seccond	74	27	27	0.06

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                phdthesis|mastersthesis|www)*>
<!ENTITY % field "author|editor|title|booktitle|pages|year|address|
                  journal|volume|number|month|url|ee|cdrom|cite|
                  publisher|note|crossref|isbn|series|school|chapter">

<!ELEMENT article       (%field;)*>
<!ATTLIST article
                        key CDATA #REQUIRED
                        reviewid CDATA #IMPLIED
                        rating CDATA #IMPLIED
                        mdate CDATA #IMPLIED
>

<!ELEMENT author    (#PCDATA)>
<!ELEMENT year      (#PCDATA)>
```

Figure 5.8. An Example DBLP DTD Extract

Linux 2.4 kernel and Oracle 9.2. The XPath query performance for Apache Xindice is also measured on a smaller set of 10,000 XML instances, due to limits of scalability of Xindice. We select article XML instances only, and non-ASCII characters are converted to ASCII characters because the Xindice cannot indicate them. Figure 5.8 shows an example of DBLP DTD in part.

For each database and each index type, we made the following measurements:

- The average time of an XPath query for approximate author name: we consider the case when an inquirer wants the partial match on the author name. For example, the query could indicate last name only, but the element of an XML instance can have a full name.

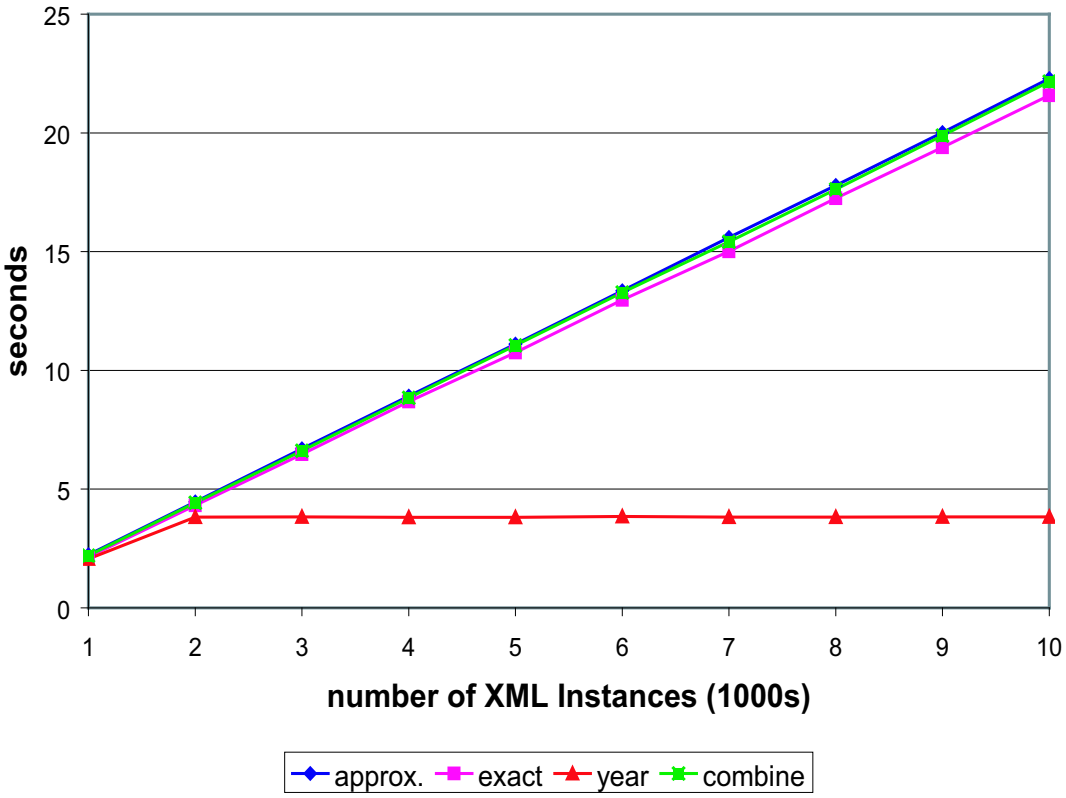

Figure 5.9. Oracle average query time with no index

- The average time of an XPath query for exact author name: in this case, the query should exactly match to the author name presented in the text element of an XML instance.
- The average time of a year XPath query: we pick "1972" for the year element match—a year that is relatively rare in the data set.
- The average time of a combined query that involves an XPath AND operation for both the author and the year

Figure 5.9, Figure 5.10, and Figure 5.11 show the average query time against XML instances from 1,000 to 10,000 rows in Oracle DBMS. We ignore the first query response because it is much slower than other queries, due to the initial object loading delay, and the

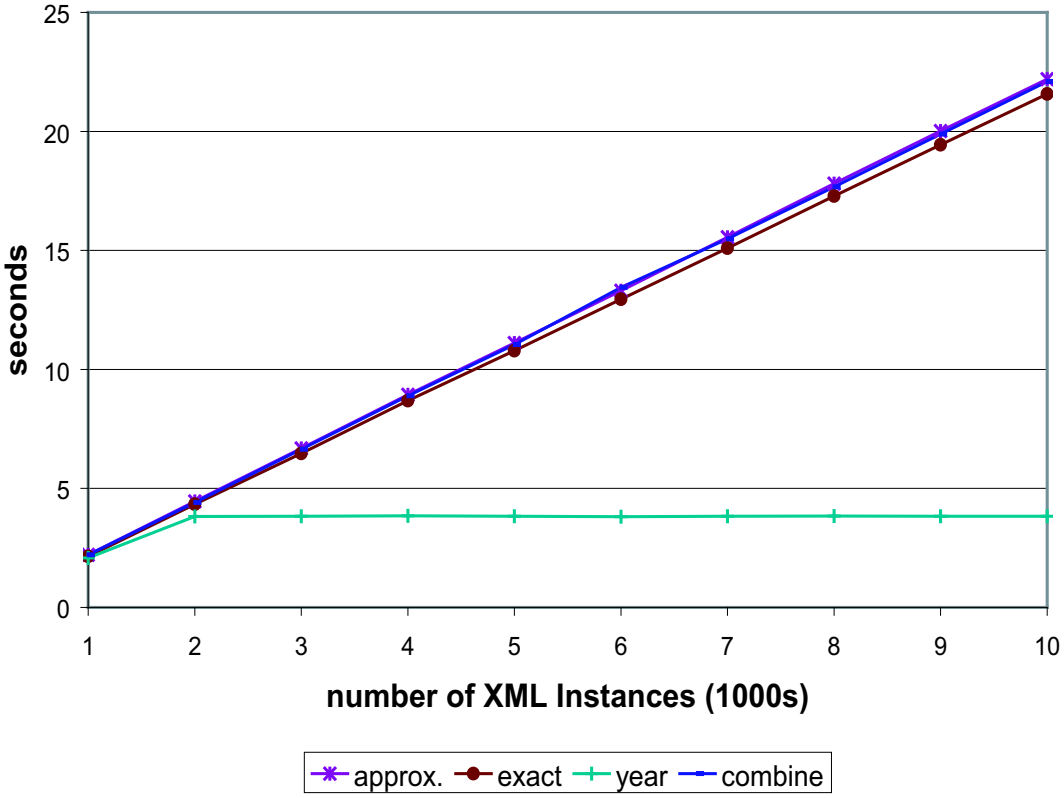

Figure 5.10. Oracle average query time with funtional index

lack of cached results in the DBMS. In Figure 5.9, the average query processing time against non-indexed and functional indexed data is increased when the size of XML instances is growing, but the year queries does not increase much from the size 2,000. The year query processing time is around 4 seconds but other queries take longer than 20 seconds for 10,000 XML instances. The context indexed cases, shown in Figure 5.11, are very fast. This implies that context indexing for XML instances is effective. Actually, many studies for the XML indexing have been based on an inverted index, and the inverted index is a basic paradigm for information retrieval. The bar graph in Figure 5.12 shows the number of query results against a year and the line graphs in Figure 5.11 are for the query processing time. The number of author query results is 1. The query time for combined author and year depends

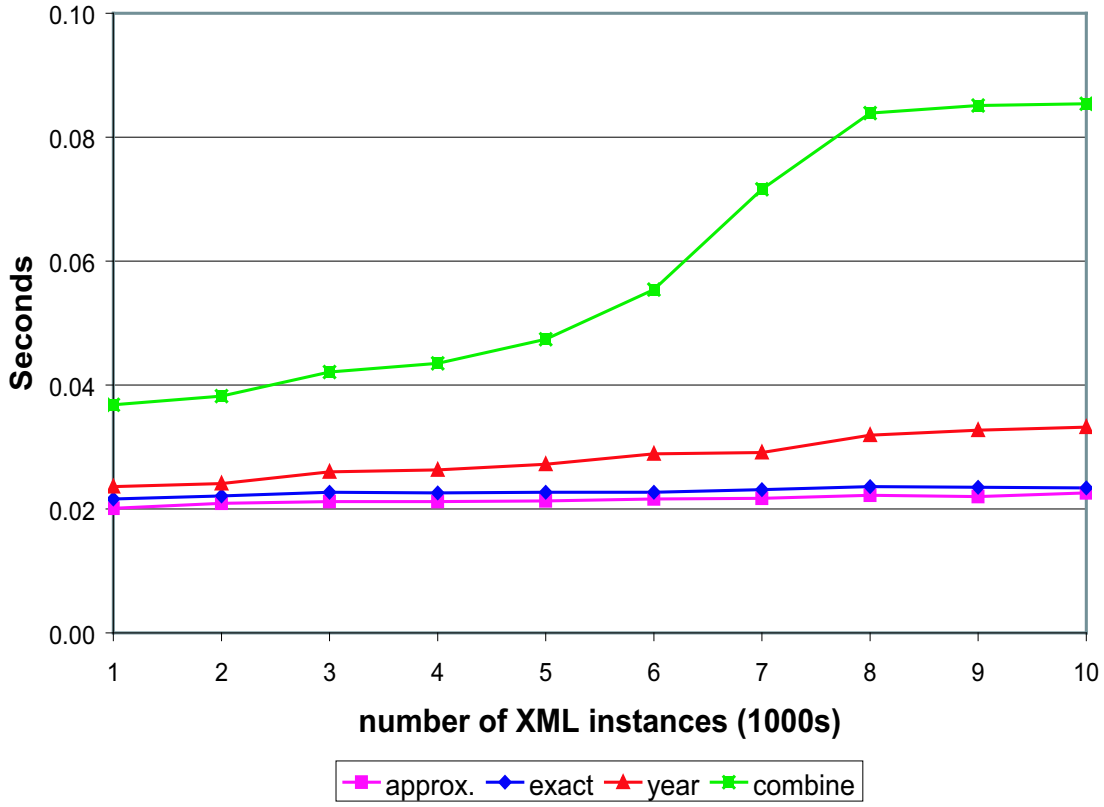

Figure 5.11. Oracle average query time with context index

on the number of year query results rather than the number of total XML instances. A possible interpretation is that the join operation dominates this combined query.

Figure 5.13 and Figure 5.14 show the average query time against 1,000 to 10,000 XML instances using the Apache Xindice database. Those graph patterns are similar to those of Oracle DBMS, but the year query time is continually increased. The query time in Xindice is longer than that in Oracle.

We also measured the query time of Xindice for numbers of XML instances between 10,000 and 20,000. From the size of 16,000, the query results were inaccurate and that should be the limit boundary of target size of XPath query for XML instances in Xindice.

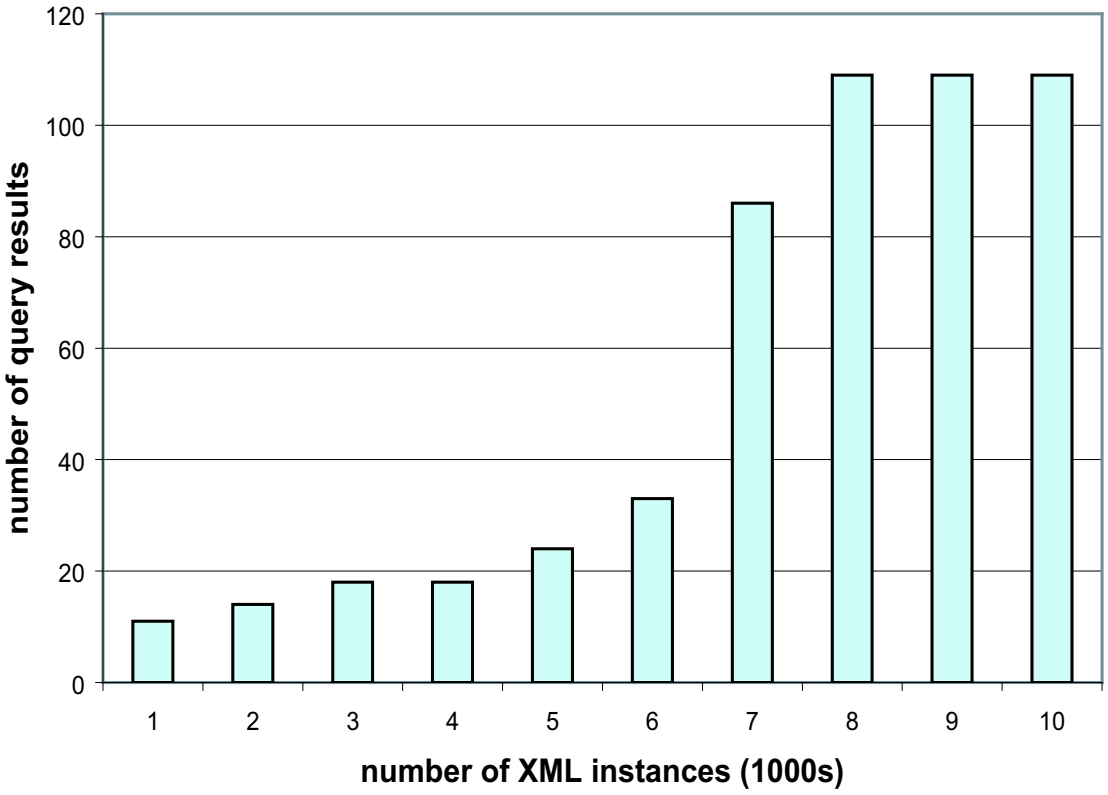

Figure 5.12. Number of year query results

Figure 5.15 shows the XPath query time for XML rows in Oracle with the context indexing, and the size of rows is from 10,000 to 100,000. The corresponding graph for number of year query results is shown in Figure 5.16.

5.6.2 Hybrid Search Query Performance

To see the cost of combining two query results, let us examine the hybrid search query processing time for a nested subquery in a relational DBMS and combining metadata and keyword search results through a hash table. For the performance evaluation of the keyword query part for unstructured data, we use 100,000 abstract text files from the TREC collection, OHSUMED [74]. We cut off embedded metadata in the collection and extract the abstract part only. The machine used is the same as in the previous section.

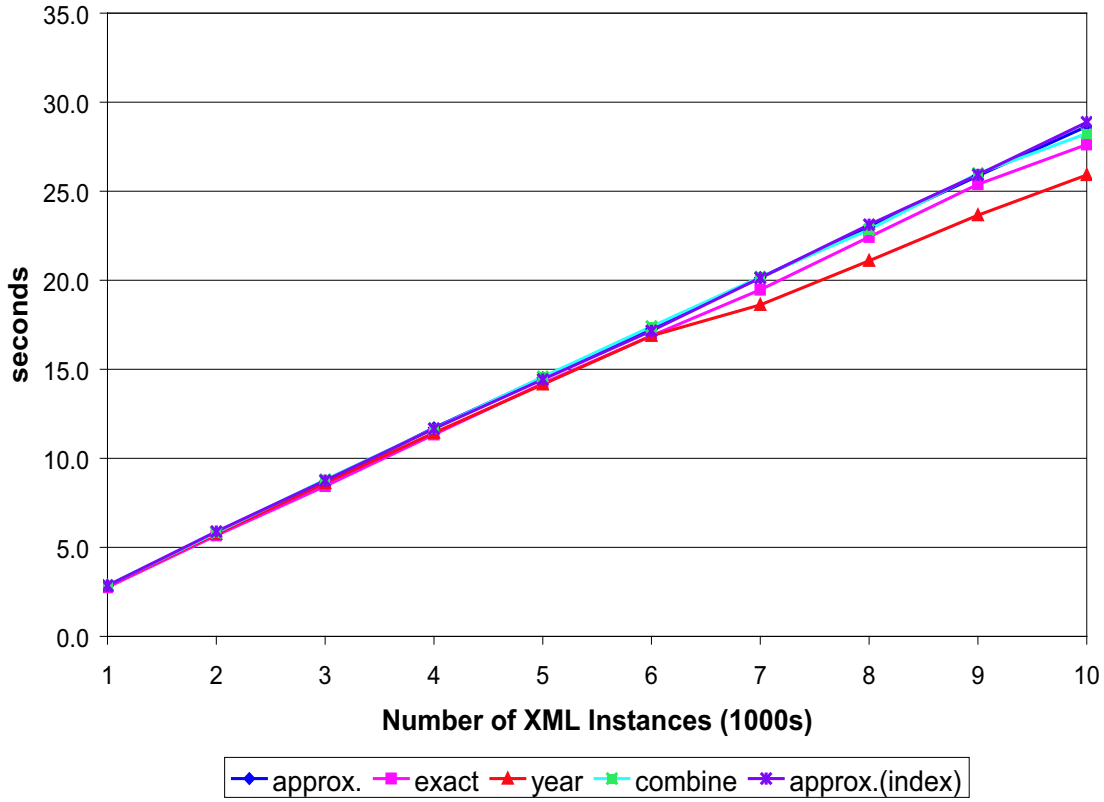

Figure 5.13. Xindice query processing time with no index

Table 5.2. Hybrid Search Query Time in Oracle

Metadata	Author	Year (Nested subquery)	Year (Hash table)
Few keywords	0.04 sec.	82.9 sec.	5.70 sec.
Many keywords	0.48 sec.	Half hour	6.96 sec.

We first evaluate a single keyword query in Oracle DBMS. We choose two keywords. One keyword is included in 4 documents and 41,889 documents contain the other keyword. An initial query takes 267 milliseconds for the former keyword query and 2,553 milliseconds for the latter. If the query is repeated, they take only 15 and 16.5 milliseconds by reason of the internal cache in DBMS. This result implies that the performance of hybrid search should depend on the performance of XML query processing or combining operations.

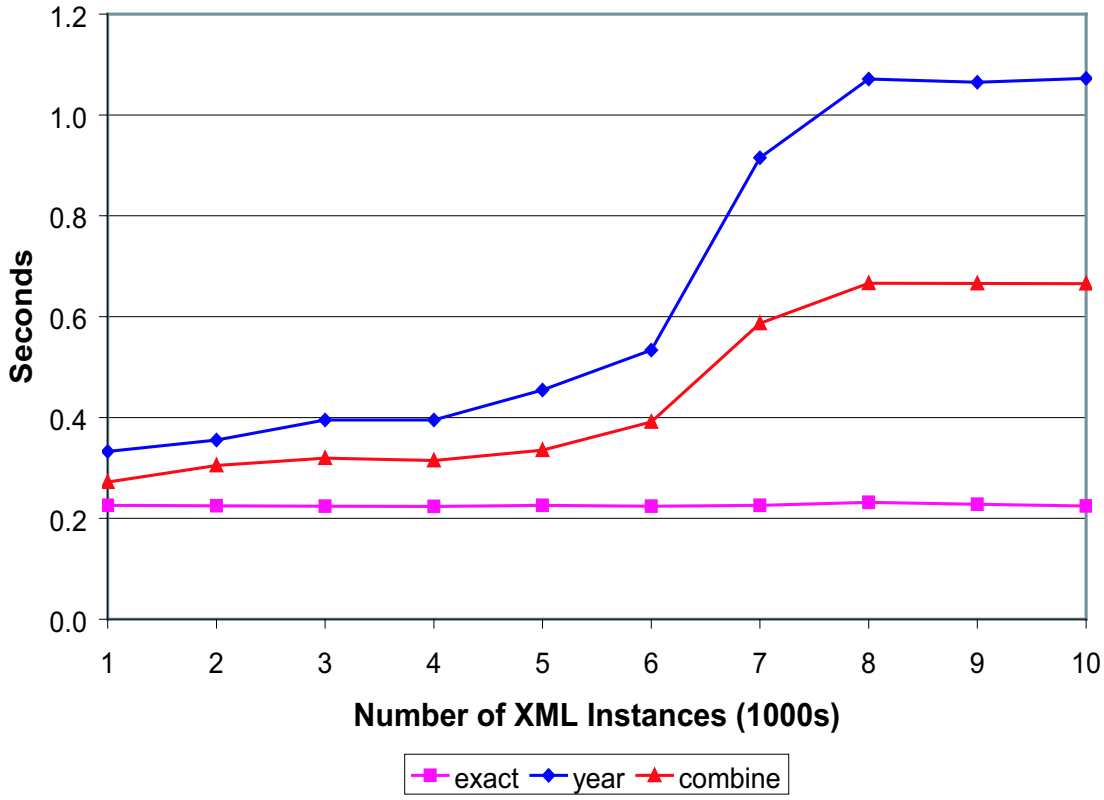

Figure 5.14. Xindice query processing time with indexing

Table 4.2 shows the hybrid search query time in Oracle DBMS. We pick an author name which exists in one element of an XML instance. The year in the query is "2002" and it is found in 7,752 out of 100,000 XML instances. The few keywords and many keywords are the same as in the keyword-only query cases in the previous paragraph. All queries are against the indexed storage. The results show that nested subqueries are much slower than using hash tables for combining many result sets in metadata. We can interpret that the joining algorithms in the relational database must be inefficient. We informally evaluate Oracle 10g in one of the cluster computers specified in the next chapter with large resource allocations. With this newer version and modified configuration, the query time of nested subqueries is less than 1 second for the many keyword matches. We also measure the time for putting the result set into the hash table, and the average is 81% of the total query time.

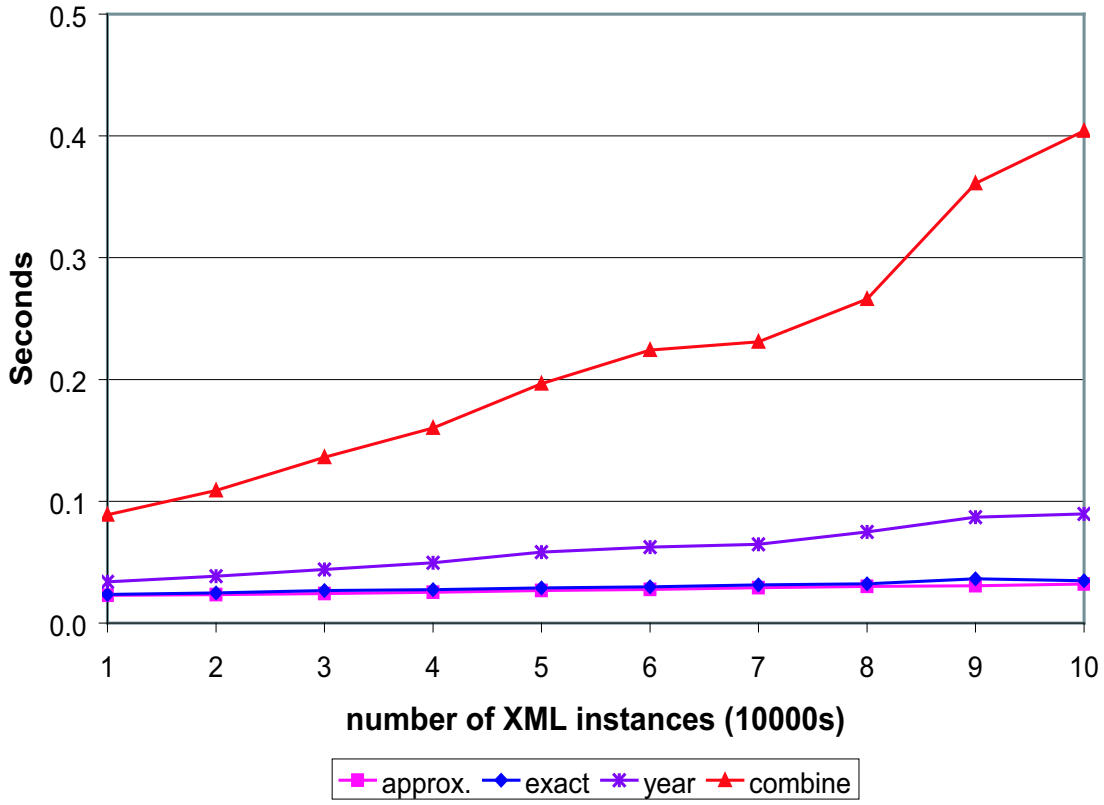

Figure 5.15. Oracle average query processing time with context index

We evaluate another keyword-only query in Jakarta Lucene. In this experiment, we only use first 10,000 entries of the document set used with the Oracle DBMS, because the Xindice XML database—the other part component for hybrid search—cannot process the query properly for more than 16,000 XML instances from the DBLP data set. Java Hotspot VM 1.4.1 is used with Jakarta Lucene 1.3 on the same machine for the measurement as in the previous section. We choose two keywords. One keyword matches in 9 documents and the other keyword is included in 4,563 documents. The initial query takes 95 milliseconds for the former keyword and 110 milliseconds for the latter keyword. From the second query, they take only 2 and 15.5 milliseconds due to the internal cache.

Figure 5.17 shows the hybrid search query performance measurement using Apache Xindice 1.1b4 and Jakarta Lucene 1.3 on the same machine. We use 10,000 XML instances

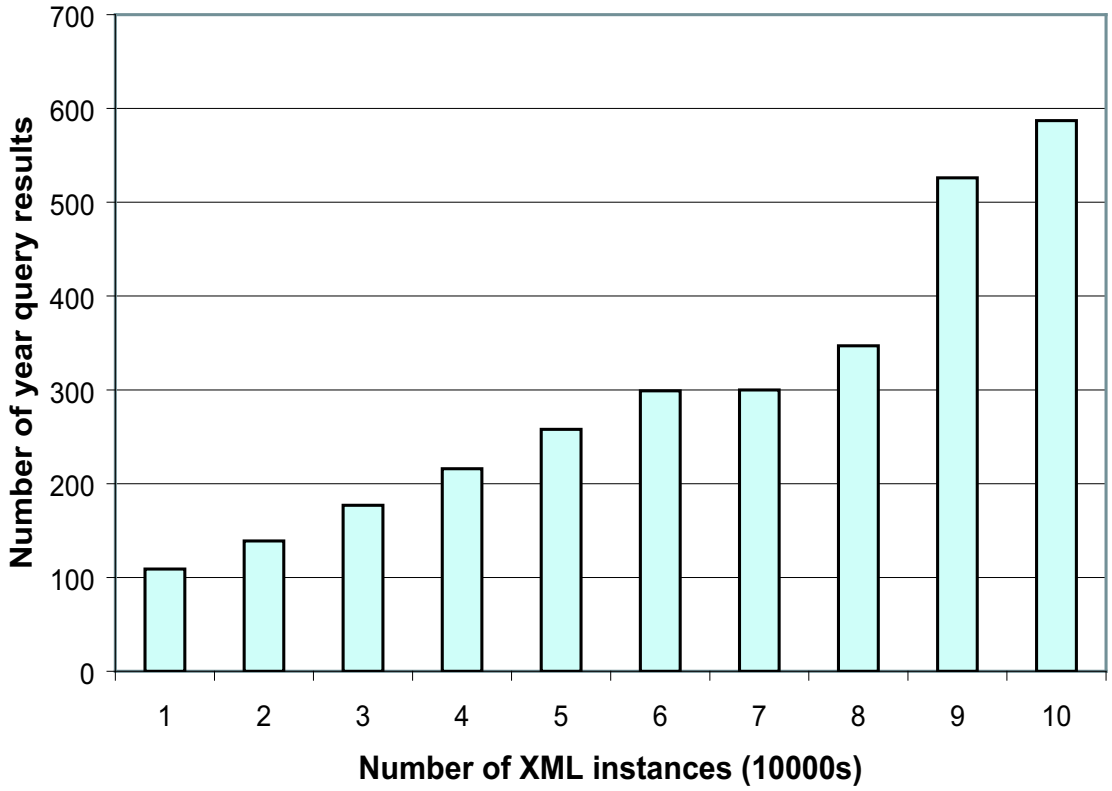

Figure 5.16. Number of year query results

and 10,000 text files that are part of the same dataset used in the Oracle DBMS performance measurement. In the figure, the final result combines year matches in 192 XML instances with keyword matches in 4,562 documents for the large data set. There are just two XML matches and a single keyword match for the small data sets. From the graph, we can interpret that XML query processing time dominates the hybrid search query processing time.

5.7 Discussion

In this chapter, we described our approaches to hybrid search at a local level. We utilized an XML-enabled relational DBMS with nested subqueries to implement the combination of query results against unstructured documents and semistructured metadata. The system

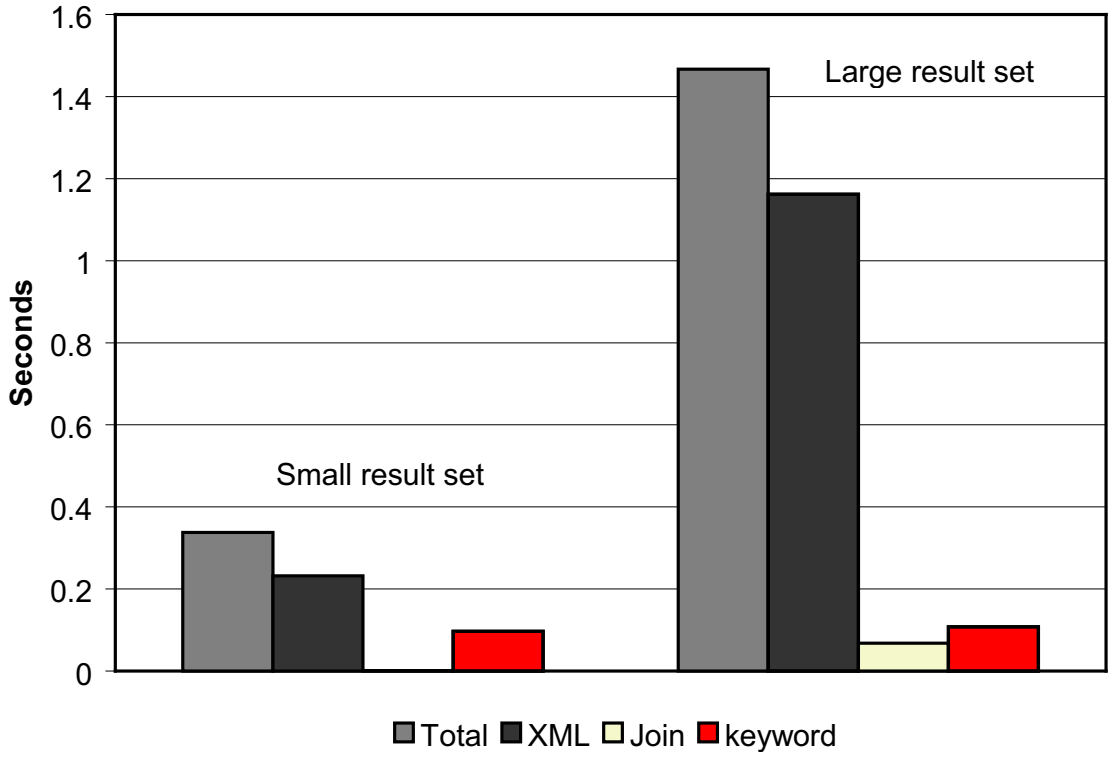

Figure 5.17. Hybrid search query processing time

integrated XML-enabled and text management features, and provides convenient SQL commands to implement the hybrid search requirements. From the experimental performance measurements, we recognized that the new Oracle Text based indexing over XML instances usually provides proper query response at the local level. However, the DBMS approach may not have appropriate query efficiency under some extreme query loads due to expensive joining operations.

Another approach to hybrid search was based on a native XML database and a text search library. As shown in the experimental measurements, our joining operation through a hash table was effective enough to process small data set with the simple data-centric metadata. However scalability is less than the system based on XML-enabled DBMS, due to limits of the native XML database.

To address such scalability problems, we will generalize hybrid search to a distributed environment and show some experiments to demonstrate the performance improvement in the next chapter.

CHAPTER 6

SCALABLE HYBRID SEARCH ON DISTRIBUTED DATABASES

The previous chapter tackled the semantic problems of the general search paradigm. We adopted a strategy that involved attaching metadata to existing unstructured data, providing the inquirer additional options to narrow the search using a hybrid approach. In the experimental performance measurements, we recognized the scalability limitations of a single-machine implementation. That chapter showed in particular that Apache Xindice had very limited scalability. In this chapter, we will adopt a distributed strategy to improve the scalability of hybrid keyword search. We will experimentally illustrate the performance improvement and the larger scalability of the new architecture.

6.1 Hybrid Keyword Search on Distributed Databases

A distributed database is a collection of several or many databases stored in different computers and *logically interrelated* through a network. A distributed database management system (DDBMS) can be defined as a software application that manages a distributed database system so that to users it seems like a single machine—it provides transparency. Let us describe our distributed architecture according to the transparency criteria considered by Ozsu and Valduriez [75].

- Data Independence: the hybrid searches on each machine are logically and physically independent of each other. The data of each database have the same schema and the schema does not change. Each database has its own management system and only returns query results against the user inquiry. So our architecture has physical data independence due to data encapsulation in each machine.

- Network Transparency: the network connection of our system depends on message-oriented middleware or peer-to-peer middleware, and the middleware administrator or agent is only concerned with the connection. The end-user does not perceive the detailed network operation of the distributed hybrid search inquiry.

- Replication Transparency: we assume our distributed architecture is restricted to a computer cluster, and no replication exists in the cluster. Simply our system is replication transparent. Data replication is usually necessary to increase the locality of data references in a distributed environment. Actually, the locality is guaranteed in a clustering architecture. In this chapter we focus on the scalability provided by the distributed environment, to address the problems exposed by our query experiments on the local machines. We will allow the replication on the peer-to-peer federated databases in a later chapter, and this replication has a different standing on a P2P overlay network.

- Fragmentation Transparency: our experiments with a hybrid search on a distributed database will show that the data can be partitioned into each database within the limitation of the local database. Full partitioning is the easiest case for the distributed database management system. The data is partitioned into a chunk of XML instances with their associated unstructured data, and this type of fragmentation is *horizontal*.

We can summarize that in our architecture each database is totally independent. The query result for the distributed databases is the collection of query results from individual database queries.

Data independence and horizontal fragmentation features make our architecture different from other general architectures for federated database systems surveyed in [76]. If the data is not independent and the user inquiries require joining query data from different machines with various schemas, we can consider a distributed querying framework based on the recently emerging Grid infrastructure [77, 78]. OGSA-DQP (Open Grid Services Architecture; Distributed Query Processing) [79] is an example of such framework. The *Open Grid Services Architecture* (OGSA) is based on a Grid system adopting Web Services [80], which provide system independent description, discovery, and invocation for machine-to-machine interactive operations on communication networks. So, OGSA has dynamic discovery and

state aware resource allocation features from Grid systems as well as other features from Web Services.

Under the OGSA-DQP architecture, distributed resources are registered in metadata format and the availability for those resources can be obtained upon request from a query service server—*Grid Distributed Query Service* (GDQS). The database schemas are accessible from the data service factory—*Grid Data Service Factory* (GDSF), which identifies data sources. The factory creates a network session for a particular database—a *Grid Database Service* (GDS) —to which a requester can submit a query. The query is evaluated and delivered to the appropriate data partitions. Those query processes are complicated and require knowledge of low-level data schemas and their relationships.

6.2 Distributed Database Architecture

The architecture of the hybrid search service on each local machine is identical to the architecture described in the previous chapter. The remaining issue for organizing the distributed database is the computer network technology that connects the databases, and provides the network transparency to the user.

Tanenbaum and Steen [59] suggested that message-oriented middleware is one of the best application tools for integrating a collection of databases into a multidatabase system. We utilize a message-oriented middleware implementation—NaradaBrokering [62, 81]. It includes JMS compliant topic-based communication, which meets the minimum requirements for the network transparency in a distributed database. NaradaBrokering also provides a cooperating broker network for increased network scalability. The detailed communication mechanism for our architecture will be described in later sections.

Figure 6.1 summarizes our general architecture. The search client is a publisher for a query topic, and a group of search services subscribe on the same topic. When the client publishes the query, the query message is broadcast to the search service subscribers. The query results from the search services are returned back to the client by publishing the message on a dynamic virtual channel—a *temporary topic* whose session object was attached to the query message originally delivered to the search service.

The architecture can be grown to a cooperating broker network as in Figure 6.2. One or more heterogeneous search services could be attached to each broker and each broker

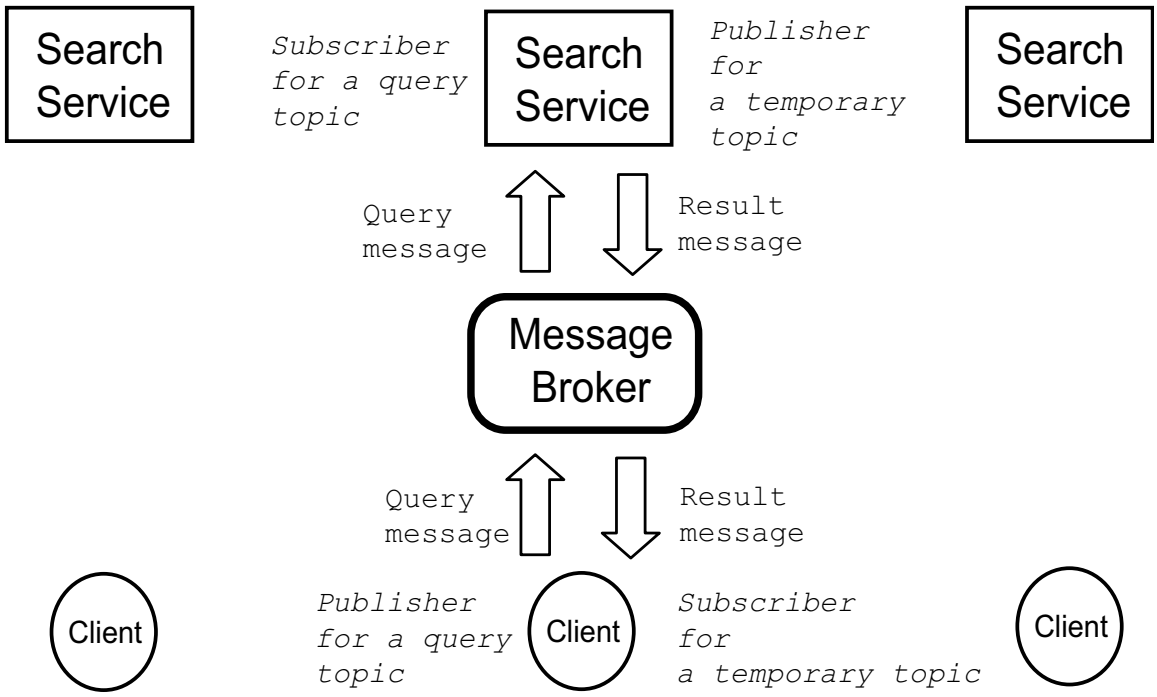

Figure 6.1. Scalable Hybrid Search Architecture on Distributed Databases

can relay messages to other brokers. The cooperative network usually follows a hierarchical structure, for better performance.

6.3 Query Processing

The query processing of each database (fragment) on our distributed architecture is identical to that at the local level described in the previous chapter. Due to the full partition in our database distribution, the query processing in the DDBMS is simple, and query propagation and result collections are the only interesting processes.

A query processing architecture on a distributed database is shown in Figure 6.3. The search client is a query publisher and the search service is a query subscriber. The user types query parameters through a user interface, and they are amalgamated into a message along with a job property and a temporary topic. The job property in this case is an integer property assigned to the "QUERY" final variable. The JMS `MapMessage` message type is

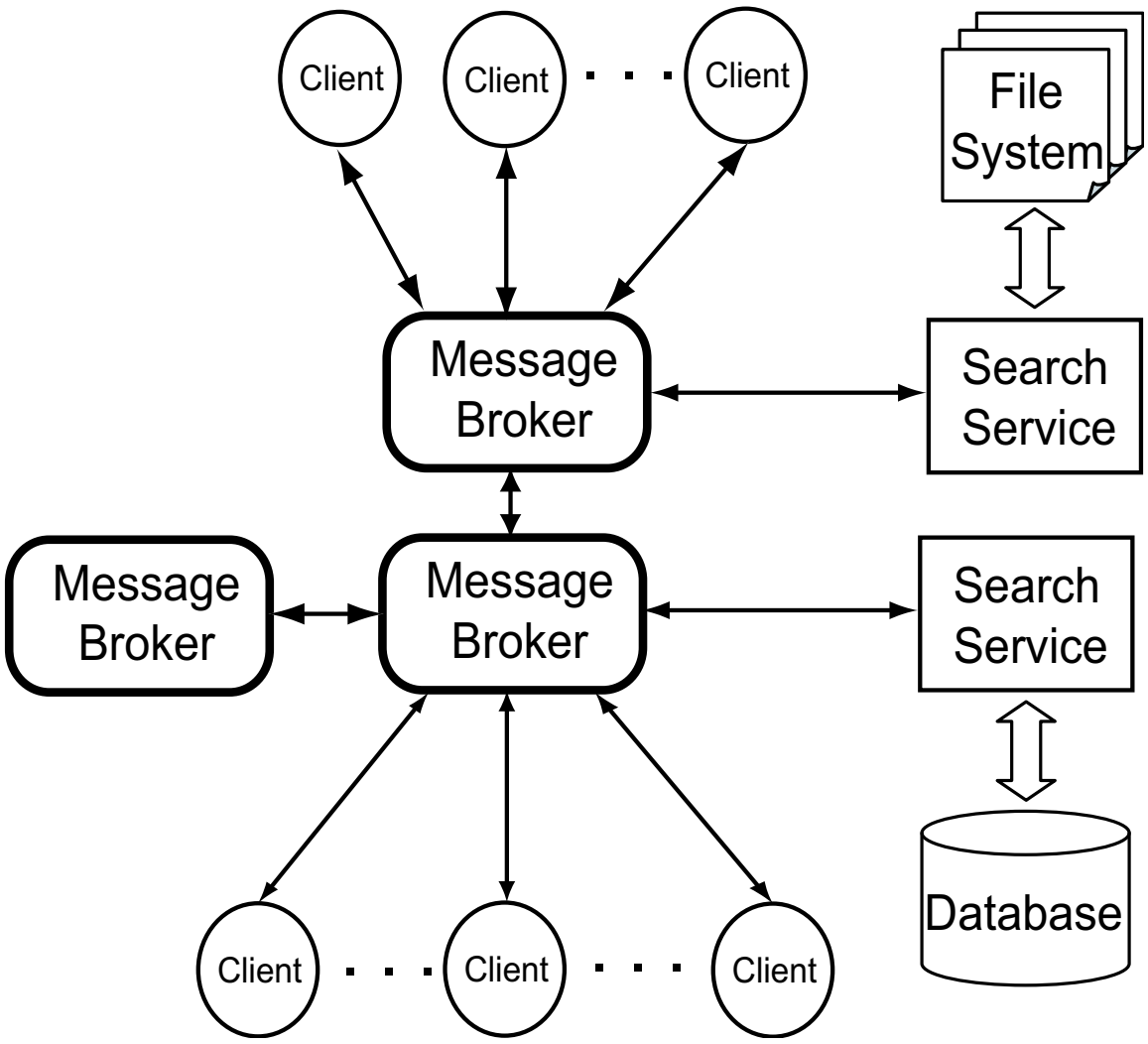

Figure 6.2. An Example of a Cooperating Broker Network

used for the query message in our Java program. The following Java fragment shows an example of a message creation, content and property setting, and message publication.

```
MapMessage message = pubSession.createMapMessage();
message.setIntProperty("job", QUERY);
message.setString("author", "an author name");
message.setString("title", "a title");
```

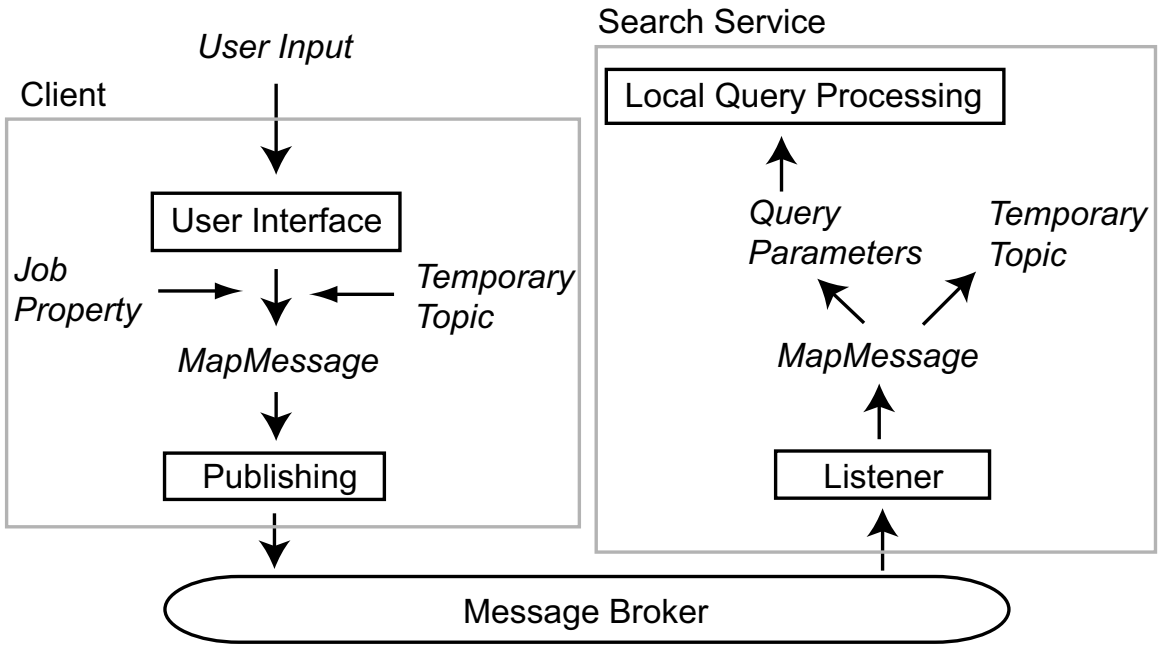

Figure 6.3. A Query Processing Architecture on a Distributed Database

```
message.setInt("year", 2004);
message.setString("source", "a reference name");
message.setString("keywords", "one or more keywords");
message.setJMSReplyTo(tempTopic);
publisher.publish(message);
```

The content of this message is a set of *name-value* pairs. The name is always a string and the value can have one of several allowed types. We used a string and an integer type in the example. Those values could be empty for the string, and zero for the integer, if there is no user input. The `JMSReplyTo` property is a message header that contains a temporary topic.

The message broker delivers query messages to search services that have already subscribed to the same topic. The listener in the search service captures the query message, which includes a property header filled with a temporary topic. The extracted query

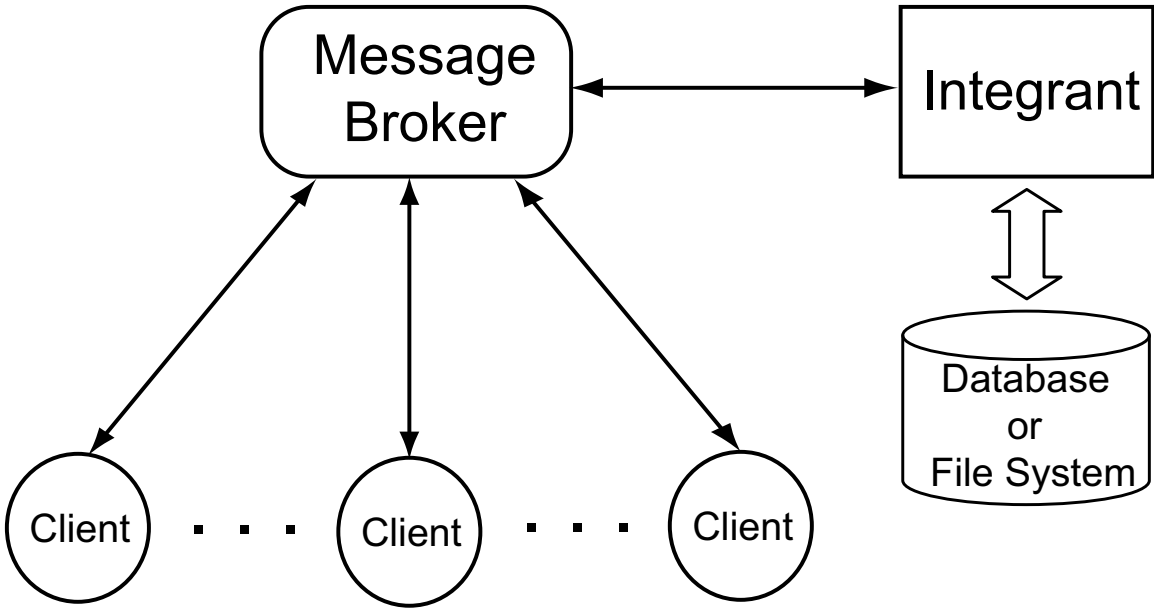

Figure 6.4. An Integration Hub Architecture

parameters are passed to the local query processing service, which produces query results as described in the previous chapter.

Those query results are returned back only to the client that published the query message. This works because the temporary topic is unique to this client. The inquirer client listens for the returned messages, and displays query results to the user.

6.4 Case Study: Data Integration Hub

In the previous sections, our assumption for the clients on our distributed database is that all of them are read-only query clients. In this section we take into account that clients may desire to be a data provider without providing an independent search service. Several or many data providers can share their information through a centralized database. They may upload, query, and download unstructured data with attached metadata via a central DBMS.

We demonstrate a data integration hub on the central DBMS using message-oriented middleware. This integration hub is an integrated version of the hybrid paper search

introduced in Section 5.3.2. The general architecture of a data integration hub is summarized in Figure 6.4. Those clients can communicate with an integrant through a message broker. As in the read-only query case, clients are publishers and the integrant is a subscriber. Clients and the integrant use the same topic. The integrant manages uploading of the metadata and unstructured data, query wrapping, and file transfer. Requests are classified by the job property, which is attached to the message sent from client to integrant. A temporary topic is also delivered to the integrant, similar to the earlier discussion.

We use different message types for each service request, as follows:

- Upload request: initially the client should request to upload to the integrant, before the unstructured data files are sent. The message for this request includes a file name of unstructured data, metadata, and a unique client user name. They are string contents in a `MapMessage` type message. The integrant checks the validity of the metadata against the XML schema as clients may provide no well-formed or valid data. The central database should avoid redundancy for the unstructured data stored in a file system. The name and directory of the unstructured data, which is generated by combining the user name and the file name, is used for a primary key.

- File upload: after an upload request is granted by the integrant, the client can send a file for the unstructured data. For the file uploading, we use a `ByteMessage` type message, which is appropriate for sending a stream of uninterpreted bytes. The client sends a header message first, and the file is sent buffer by buffer through message publishing. A job property, a file name, a user name, and a temporary topic are included in the header message. A JMS message has a unique ID when it is created, and the ID is attached to each message. The integrant can classify those file upload messages by extracting the message ID. This mechanism is necessary because several file uploads can occur simultaneously.

- Query request: query message communication has the same mechanism as in the read-only query described before. A `MapMessage` type message, which includes query parameters and properties, is published to the integrant. The query results are delivered back to the inquiry client subscribed to a temporary topic.

- Download request: the target unstructured data in a file can be obtained from the query request. The client subsequently requests a file download with a `MapMessage` type message that includes the file name and a temporary topic. The listener on the client then captures the `ByteMessage` type message published from the integrant, and the target file is written message by message on the client machine. Each message includes the file name property. The message broker is responsible for preserving the order of message transfer.

The database schema of our data integration hub is similar to those in the hybrid paper search in Figure 5.3, but there is an additional table for the file uploading for the unstructured data—a temporary file locator table. Our system allows the file upload on a temporary directory only, and moves the files to the designated directory later. When a user requests a file upload and incidentally the same file name already exists, a new file name is assigned for the final destination by the integrant. The original file name is stored in the table for metadata, but the actual file name is stored in the unstructured data table. We assume that a user does not assign the same file name to two or more different set of unstructured data. A naming and directory for each row in the paper metadata table is generated from combining the unique user name and the file name, and it makes the naming and directory to a potential primary key.

Another aspect for the integration hub, which was not introduced in the local hybrid paper search, is the metadata validation against XML schema. The storage for the local hybrid paper search is managed by database administrators, but ordinary users can upload their own data to the database of the integration hub. We utilize an Oracle DB operation to check the validity of metadata presented in XML instances against a registered XML schema. The XML schema for our hybrid paper search is shown in Figure 6.5.

Each data integration hub has a message broker and an integrant. A group of data integration hubs may provide a global search over a distributed information system by using the cooperative network features built in to NaradaBrokering, or by using an additional network layer—a peer-to-peer overlay network.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="bibliography">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="authors">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="author" type="xs:string"
                maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="source" type="xs:string"/>
        <xs:element name="year" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 6.5. An XML Schema Example for Hybrid Paper Search

6.5 Experimental Performance

In this section, we evaluate our hybrid search over a distributed database. A cluster of 8 computers forms a distributed environment, and all computers are located within a local network whose network bandwidth is very high.

In these experiments we use the same 100,000 XML instances from DBLP [73] and 100,000 text files from OHSUMED [74] as the hybrid search experiments in the previous chapter. The data set is horizontally partitioned into 8 fragments, and each fragment on each machine has 12,500 XML instances and 12,500 text files. All eight computers have the same specification—2.4 GHz Intel Xeon CPU with 2 GB of memory, running a Linux 2.4 kernel and Java Hotspot VM 1.4.2. Those machines are connected with each other by 1 Gbps links, but the switch for the outside connections has 100 Mbps bandwidth. Apache

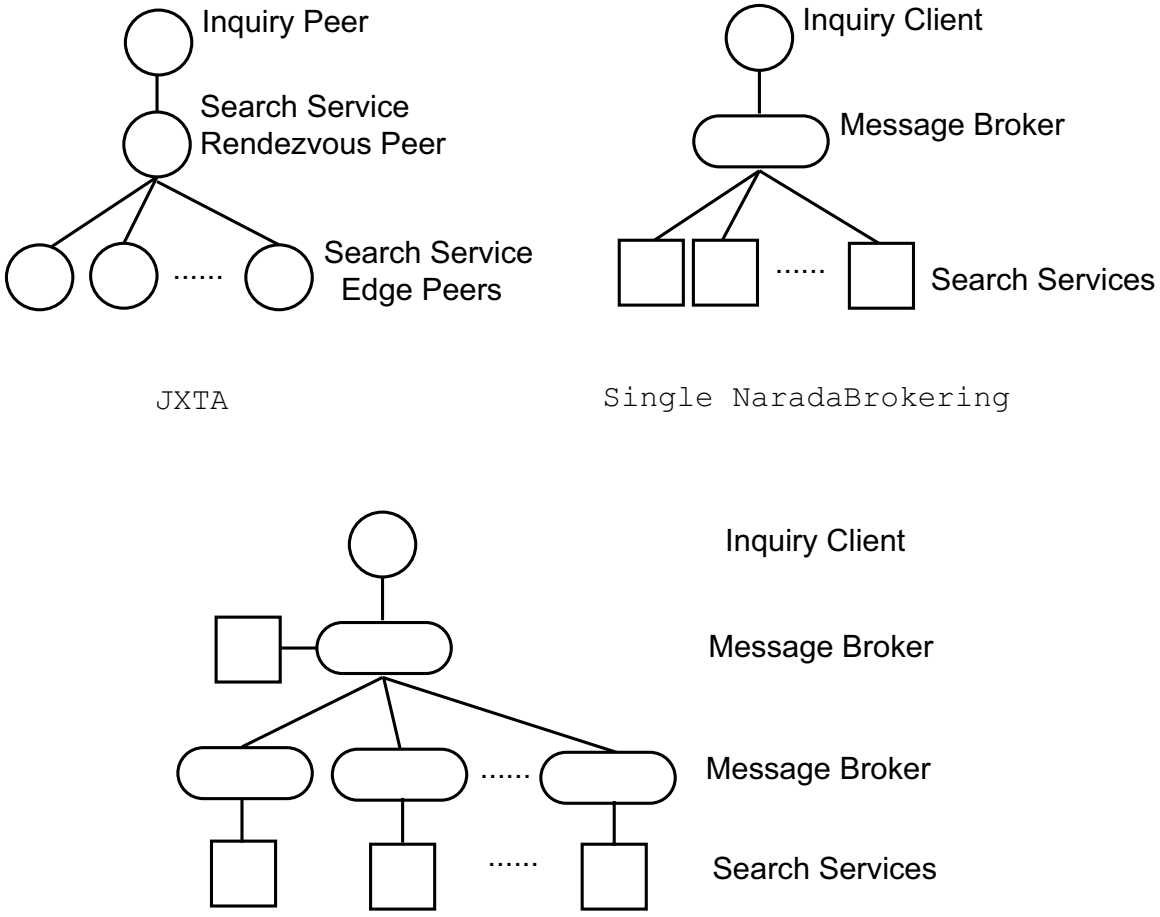

Figure 6.6. Examples of communication middleware architectures

Xindice 1.1b4 is used as a native XML database, and Jakarta Lucene 1.3 is utilized for text management.

We use two different communication middlewares—JXTA version 2.3 and NaradaBrokering version 0.96rc2—for the performance comparison, and three experimental architectures for 8 nodes as follows:

- JXTA: 1 node acts a rendezvous node and all other 7 nodes are connected to the rendezvous node. All of the nodes are located in the same subnet and all the query propagations are broadcast only within the subnet. This is because the current JXTA

implementation does not meet the specification for selective query propagation from the rendezvous peer. The detailed paradigm and problems of JXTA are described in a later chapter.

- Single NaradaBrokering: 1 node has a server and a search service, and the other nodes only have search services. All the search services are clients for the communication.
- Multiple NaradaBrokering Cluster: 1 node has a root server and the other nodes have second-level servers. Each node has a NaradaBrokering server and a search service. This architecture gives us an idea of the performance difference between a cooperative network and a single message broker.

Examples of these architectures are shown in Figure 6.6.

Figure 6.7 shows the average response time for an author exact match query over 8 search services connected using the three approaches. We choose 8 queries that combine an author and a keyword, and the databases are indexed against the author element in the XML instances and keywords for the unstructured data. Each query matches to only one of eight search services. The number of matches is between 1 and 3. We present the graphs separately for the average response time of no match and match result cases. We can interpret the difference between matched and non-matched query time to mean that the additional overhead for processing matched results is more than half of the total query time. The local processing time for non-matched query is very short as a result of the indexing against the author name. But the matched query takes a longer time for joining the query results, and it is larger than the communication time. The query time of NaradaBrokering connections is shorter than that of JXTA connections. The time for eight connections of NaradaBrokering is a little shorter than 1 broker connection. But considering standard deviations—more than 70 ms—the two NaradaBrokering cases are statistically indistinguishable. NaradaBrokering uses Java NIO—the new I/O for better performance provided in JDK 1.4. JXTA version 2.3 does not use the Java NIO features. We enable the NIO option in the NaradaBrokering performance measurement.

We evaluate further hybrid search queries for the year match. In these queries, there are two different keyword selections. One has a few keyword matches—only 4 documents in 100,000 unstructured data documents, and the other has many keyword matches—41,889

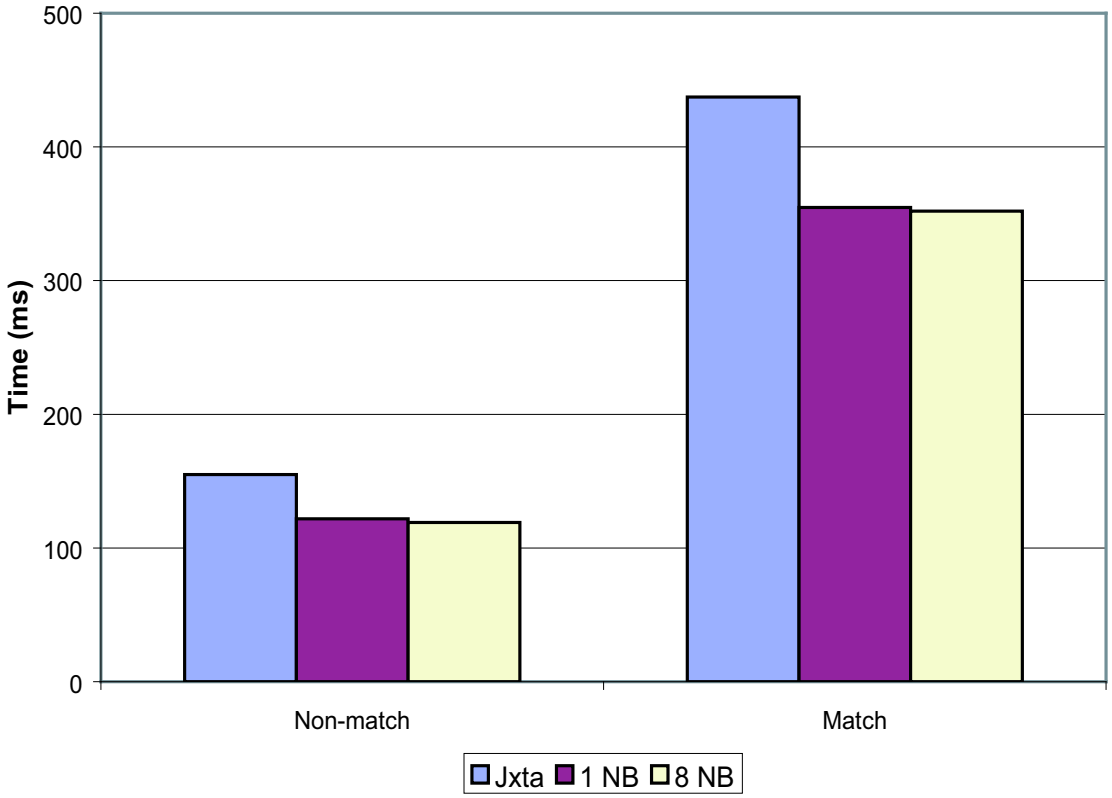

Figure 6.7. Average response time for an author exact match query over 8 search services

documents out of 100,000. The year in the query is "2002" and it is hit in 7,752 out of 100,000 XML instances. These selections are the same as those in Table 5.2. We only show a single NaradaBrokering case because multiple NaradaBrokering cluster had little difference. Figure 6.8 shows the average response time for the year match hybrid query over 8 search services in the cluster. Compared with the performance experiment results on a single machine shown in Table 5.2, the year match queries on the distributed database show dramatic improvement in performance. The query time is improved from 82.9 seconds to less than 0.3 seconds for a few keyword matches, and from half an hour to less than 0.3 seconds for many keyword matches. This big performance improvement derives from *horizontal partitioning*, which reduces data retrieving time from disk with the relation partitioning. We divide those data into each machine by continuous ranges, and it is called *range partitioning*. The second

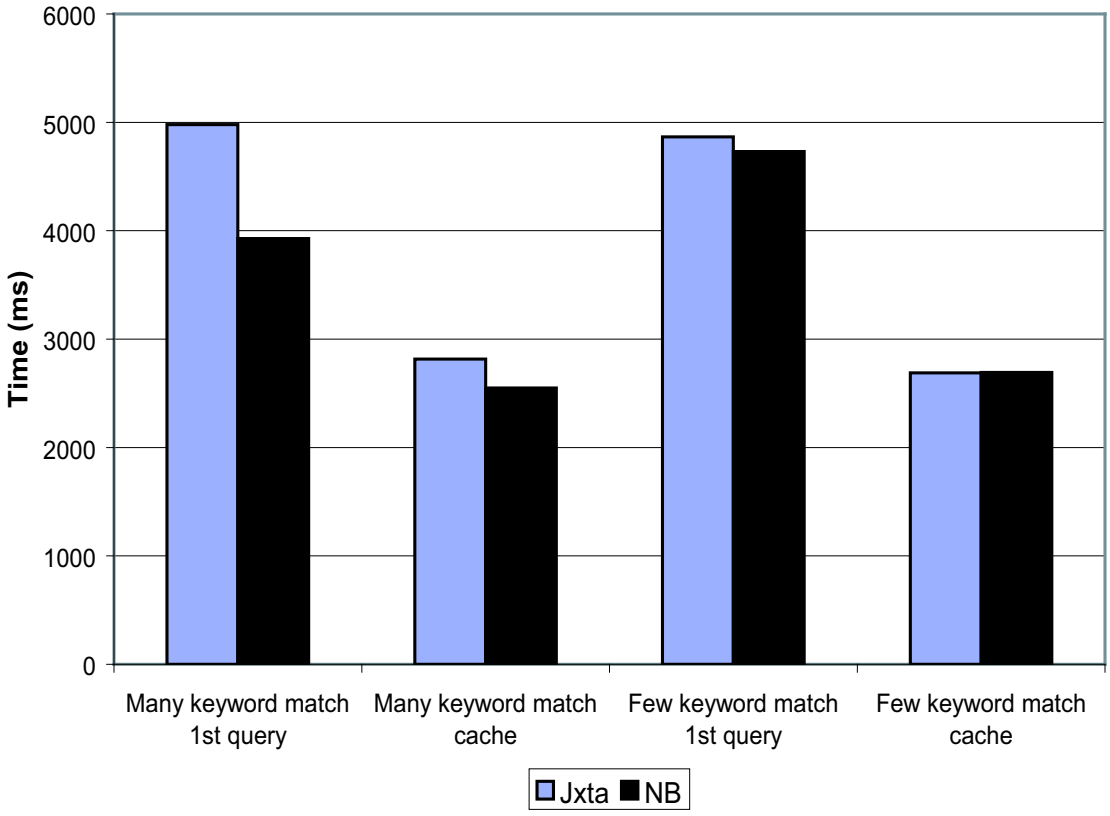

Figure 6.8. Average response time for a year match query over 8 search services

and later repeats of the same inquiry take a shorter time than the first attempt. This is because the internal cache on the databases improves the query response time for recently answered inquiries. The query time in NaradaBrokering middleware is faster than that in JXTA, similar to the author matches.

6.6 Discussion

Starting from the local hybrid search in the previous chapter, we developed a scalable hybrid search using a distributed environment. Those distributed architectures are mainly based on several or many computer network connections utilizing a message-oriented middleware or a peer-to-peer network framework. We also described a data integration hub,

which is another application communicating through the message broker with a centralized control. The integration hub can be a search service node in a distributed database, or a peer in a peer-to-peer overlay network. This will be discussed further in the next chapter.

Through experimental performance tests, we established that a distributed database can overcome the limit in target size of XML query when Xindice is run on a single machine. There was no problem up to 100,000 target XML instances in the query over the distributed database. The local native XML database could not produce accurate query results when the number of stored XML instances was over 16,000. Another contribution from the distributed query processing is the significant performance improvement for some queries over large target results. Our experiment was focused on the exact match, because Xindice does not provide context-based indexing on XML elements and attributes—there was no performance improvement from XML element indexing for an approximate match search.

In this chapter we described a scalable hybrid search on distributed databases. It provides a total query result generated from a union of queries against data fragments in a computer cluster. The aspect of horizontal partitioning for our architecture contributed a significant performance improvement for some queries comparing to those on a single machine. Furthermore our new architecture extended the scalability of Xindice XML query, limited to a small size on a single machine. We also demonstrated a data integration hub, which acts as a search service unit on a distributed environment. In the next chapter we will generalize the hybrid search to a more scalable domain that may instead provide partial query results.

CHAPTER 7

A HYBRID PEER-TO-PEER KEYWORD SEARCH

With the popularity of computer communication networks, there has been much research on sharing and exchanging information between computers, with various approaches. The emergence of the Internet motivates users to find useful information from widely dispersed resources, and the search for information on the Internet has gained its moment in the spotlight. There are two main approaches for search on the Net—searching over structured data or unstructured data. An archetypal example representing structured data is a relational database, while information retrieval represents search over the unstructured data. Extending these approaches to the Internet environment uncovers new research areas. The theories of SQL queries over structured data give a way to XML queries—searching over semistructured data—while newly developed Web search engines are based on information retrieval technologies.

Web search engines visit as many Web pages as they can collect, and provide search results against keyword based inquiry. One problem of the Web search engines is that they cannot visit every connected resource, but the bigger problem is that their query results are often totally unrelated.

The Semantic Web [2, 3] is a superlative extension of the Web. It provides multiple relation links with directed labeled graphs, and machines like Web crawlers can understand the relationship between different resources. On the other hand the ordinary Web has the hyperlink—a single relationship—and a machine cannot infer further meaning. To present the relations of the objects on the Semantic Web, the object terms should be defined under a specific domain description—an ontology. Designing an ontology usually requires a domain expert due to the sophisticated definition. Currently, most Web pages include no such semantic content, and no unified agreement of general semantic definition exists.

From search over the structured data approach, keyword search in databases [4, 5] is suggested for the Internet environment. Both Web integration with legacy database management system, and dynamic Web publication through the embedded databases, strongly benefit from keyword search capability on the databases. This makes the utilization of the database simple for ordinary users, because it does not require understanding of the database schema. Ironically, this simplification of keyword search on databases usually sacrifices the inherent meaning from the structured data.

In this chapter, we will generalize our hybrid architecture to more scalable and potentially partial match queries over a peer-to-peer overlay network. This generalization may give an intermediate search paradigm on the Internet—providing semantic values through XML metadata that are much simpler than those of the Semantic Web. The scalability cannot match the famous search engines, but our search paradigm on peer-to-peer network environment can provide a search on an exclusive overlay network. Peers in a mutual interest group can share their information and search through a customized overlay network on top of the Internet.

7.1 Peer-to-Peer Systems vs. Distributed Database Systems

We described our hybrid search on distributed databases in the previous chapter. There are some peculiar features in the P2P systems comparing to distributed database systems [82]. These include:

- Self-administrating: each node in P2P systems can control itself to forward the message to the next node, taking into account load balancing and routing. Without proper load balancing and routing, message propagations can cause message flooding in the communication network. Some initial P2P systems like Gnutella [50] depend on message flooding for their message propagation with minimal Time-to-live (TTL) parameter.

- Dynamism: nodes in P2P systems may often join and leave the network. So the domain for the overlay network is dynamic. On the other hand distributed database systems manage participation or secession of nodes.

- Schema-less: many P2P systems do not share a global schema. Their query systems usually depend on keywords. Our hybrid search system may not be one of them. The hybrid search requires semistructured metadata, but it is not restricted to a unified schema. If there is no matched metadata attribute, all that happens is that no result is produced. Distributed databases usually share some schemas, and their queries sometimes organize a joining view from several known schemas.

- Incompleteness: nodes in P2P systems may not have all of the data, and some nodes may even disconnect from the network. Therefore, queries on P2P systems may produce partial results. They are similar to those in Web search engines, which provide partial results from subset of the Internet. Meanwhile, queries over distributed databases provide results from complete data.

- Decentralization: P2P systems do not usually maintain a central host to map data to nodes directly. A query from a node is consequently forwarded to the relevant nodes among its neighbor nodes using an overlay network.

In this chapter, we demonstrate our hybrid search on a P2P system. The scope of research does not include the development of ad hoc networks, and our communication layer depends on an existing P2P communication middleware—JXTA [10].

7.2 P2P Framework

The fundamental paradigm for the hybrid search service also applies to that on the P2P systems. Each hybrid search unit has its own local query processing and repositories for unstructured and semistructured data. The data are independent as in hybrid search services on distributed databases, but data replication may exist in a P2P communication environment because there is no centralized control. Figure 7.1 summarizes the structure of a search framework unit in each search service peer. The following modules constitute a search service peer.

- Repositories for unstructured data and their metadata: the search target data and their metadata is stored and managed by the search service peer. Indexing of the data is used for efficiency. We demonstrated those repositories in the earlier chapter—one

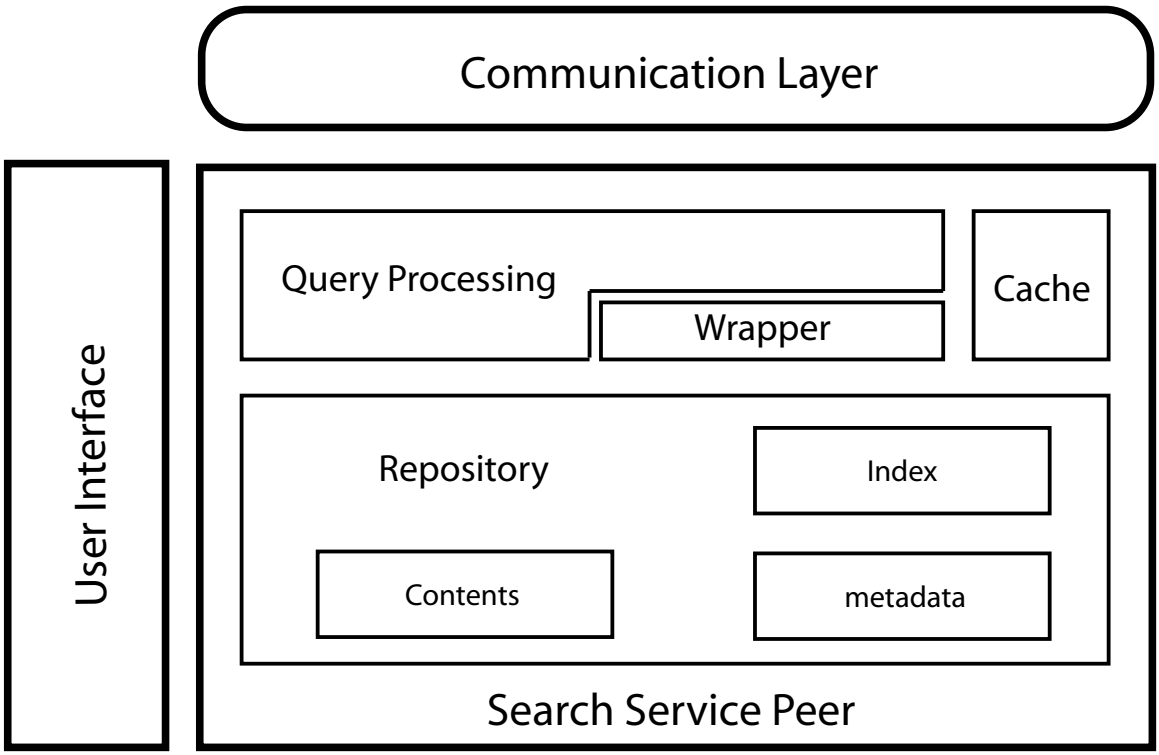

Figure 7.1. The Structure of a Search Framework Unit

is based on a relational DBMS and the other comprises a native XML database and a text search library. We utilize indexing included in those databases and library.

- Query processing for the repository and a wrapper: the query processing module extracts the target result against the user inquiries. Most database management systems provide query processing as a primary module for their repository retrieval. We described our local query processing against the hybrid keyword inquiry in the earlier chapter. Some internal codes function as a wrapper in our experiments. Therefore, they convert query messages to actual SQL queries.

- Cache: the information advertisement cache in the communication middleware, the result data set cache in the database systems, and internal cache in the repositories increase the query performance. We do not introduce any external cache in our

experiments, but utilize the caches in the database systems and the communication middleware.

- Communication layer: each search service peer communicates to share its data with other service peers through communication layer for the peer-to-peer overlay network. Our communication experiment depends on a particular peer-to-peer communication middleware.

The detailed mechanism for the query propagation and group organization on the P2P network will be described in the next section.

7.3 Peer Group Communication

A peer group is a specific aspect of the JXTA communication middleware, and it is comparable to topic-based communication in Message-Oriented Middleware (MOM). Peers in a peer group have a unique group ID and share a common set of services mutually within the group. A peer may join multiple peer groups at the same time. Peer groups could be formed over physical network barriers like different subnets, firewall domains, and Network Access Translation (NAT) boundaries. Groups can provide a hierarchical group relationship and each peer group has one parent. The root group is the Net Peer Group, which provides core services. Every peer joins this group when it initializes.

There are several kinds of peers in JXTA. A regular peer is called an edge peer, and it does not forward any query request in later version of JXTA 2.0 [58]. The rendezvous peer is a super-peer, which is usually a more powerful peer. Its network connection and availability is expected to be stable, and it plays a role of information cache for the connected peers. It connects directly to other rendezvous peers, and it can forward query requests to them. The rendezvous peer network is the backbone of JXTA communication network, and edge peers connect to it directly or through intermediate peers like relay peers. Relay peers bridge peers separated by NAT or firewalls to the rendezvous network using routing information. Any peer in a group can turn into a rendezvous peer if it has right credential.

The general sequence of the typical peer operations is described well in [83]. Let us illustrate the sequence of our group peer operations. One of the group peers that have the search service is a rendezvous peer, and its operations are as follows:

- Starting JXTA: a peer should instantiate the JXTA platform to start. The default platform object is initiated and a peer group object is created. The peer group object including the Net Peer Group—an initial group to join—is returned. This object provides the default reference implementation of core JXTA services like Discovery Service, Membership Service, and Rendezvous Service. In our experiments, we invoke a discovery service for the Net Peer Group.

- Creating a group: there are several ways to create a group, and we create it with the Module Implementation Advertisement parameter. A JXTA module is a platform-independent abstraction that provides some distributable functions. The Module Implementation Advertisement is a metadata that describes a specific implementation of a module specification. This creation method is better than creating from the advertisement, because we do not have to check whether the implementation advertisement is published before instantiating the group. The advertisement of a new group is subsequently published in a local advertisement cache.

- Joining a group: before joining the new group, the rendezvous service is started, a new credential is made, a membership service is called, and the membership is applied using the credential. Additional operations are necessary for a secure group, but we did not experiment with this case.

- Publishing a pipe advertisement: a pipe service is called, and its advertisement is published locally. Publishing puts the advertisement on the cache so that other edge peers can get the information about the pipe availability. The type for input pipe advertisement is propagational, and the type for the output pipe advertisement is unidirectional in our case.

- Opening an input pipe: an input pipe is created from a pipe advertisement. The input pipe is an endpoint of unidirectional communications. The user query reaches this input pipe. The listener catches those messages arrived into the input pipe. Pipe advertisements are reusable in JXTA.

- Opening the output pipe: this output pipe is a route back to the inquirer. Through this pipe, the query result is delivered. The output pipe should be used after binding to

the client edge peer endpoint—the opened input pipe from the client edge peer in our case. A unidirectional output advertisement is made and published, and the output pipe is created from the advertisement. A string message for the result is sent through this pipe.

Operations of a search service edge peer are as follows:

- Starting JXTA: this operation sequence is the same as that of the search service rendezvous peer up to the discovery service call. A rendezvous service is invoked and the operation waits until the rendezvous service connected subsequently after the discovery service call.

- Joining a group: an edge peer should discover the peer group advertisement from the rendezvous peer first to join. A discovery listener catches the target group advertisement if the group name is matched, and joins the group using the advertisement. Other sequence of this operation is the same as the rendezvous peer case.

- Publishing a pipe advertisement, opening an input pipe, and opening the output pipe are the same as those of the rendezvous peer.

Operations of a client edge peer are as follows:

- Starting JXTA and joining a group are the same as those of the search service edge peer.

- Publishing a pipe advertisement: this operation is similar to that of other edge peers, but the input pipe advertisement has unidirectional type and the output pipe advertisement has propagational type.

- Opening an input pipe and an output pipe: these operations are similar to those of other edge peers, but the pipe communication types are reversed.

Once the connection environment is established, we can illustrate the ordinary query and its results as in Figure 7.2. A search client peer sends a query message through an output pipe, and it propagates to the input pipes of search service peers, whose advertisements are the same as that of the input pipe advertisement in the client peer. The parameters in

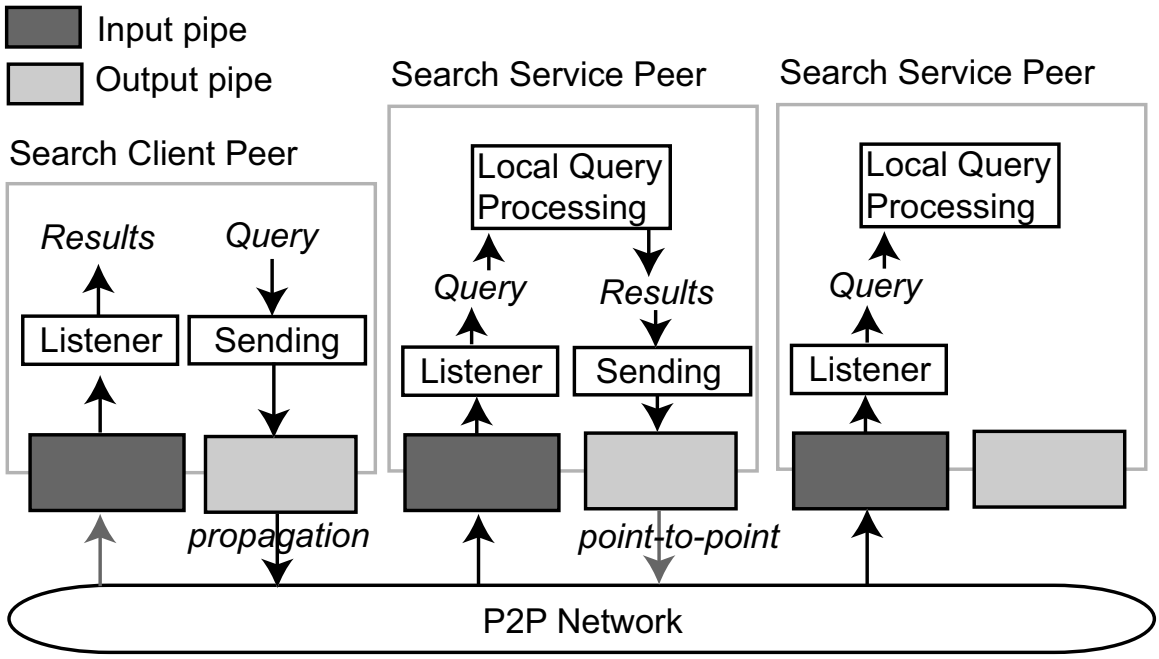

Figure 7.2. A Query Propagate and Results back on a P2P Network

the query message convert to a local query, and the local query processing produces a query result. The query message includes the pipe ID in the input pipe advertisement of the search client peer, and this unique pipe ID is used as the pipe ID in the output pipe advertisement of the search service peer. Therefore the query result is returned back only to the client peer that sent the query message. The result message transfers through a point-to-point pipe.

The query message is similar to that in the message broker described in Section 6.3. An ordered list of message elements constitutes a message, and a message element consists of a *name-value* pair. Instead of adding a temporary topic to a message like in the message broker, the unique pipe ID in a message indicates the destination for the query results returned to the peer in JXTA P2P networks. Figure 7.3 Java fragment shows an example of a JXTA message setting used in the experimental performance test in the previous chapter.

```
Message msg = new Message();
StringMessageElement sme =
    new StringMessageElement("PipeURIMsg", pipeID.toString(), null);
msg.addMessageElement(null, sme);
sme = new StringMessageElement("KeywordTAG",
                               "one or more keywords", null);
msg.addMessageElement(null, sme);
sme = new StringMessageElement("AuthorTAG", "an author name", null);
msg.addMessageElement(null, sme);
sme = new StringMessageElement("YearTAG", "2004", null);
msg.addMessageElement(null, sme);
outputpipe.send(msg);
```

Figure 7.3. An Example of a JXTA Message Setting

7.4 Experimental Performance for Peer Group Communication

In this section, we evaluate the peer group communication performance in JXTA P2P networks. We have already evaluated the local query processing time for our hybrid search in the earlier chapter, and the remaining major performance factor for the hybrid search on P2P networks is message response time from a query client to search service peers. In particular, we focus on the time from sending a propagation message to receiving an answer through a point-to-point pipe. Detailed performance tests including other part of JXTA operations, which were described in the previous section, can be found in [84].

Figure 7.4 shows our experimental setup for measurement of the message response time. We have 3 different subnets available. The query client is located in subnet A, a subnet B includes a rendezvous peer that is a JXTA shell, and one group rendezvous peer and the other 7 edge peers belong to subnet C. The subnet A and C are 4 hops apart, the subnet B and C are 2 hops apart, and the subnet A and B are 4 hops apart.

The client machine in the subnet A has 1.2 GHz Sun UltraSPARC-III CPU with 16 GB of memory, running a SunOS 5.9 and Java Hotspot VM 1.4.2. The specification of the machine in the subnet B is 460 MHz Sun UltraSPARC-II CPU with 1 GB of memory, running a SunOS 5.8 and Java Hotspot VM 1.4.1. All eight computers in the subnet C have the same specification—2.4 GHz Intel Xeon CPU with 2 GB of memory, running a Linux 2.4 kernel

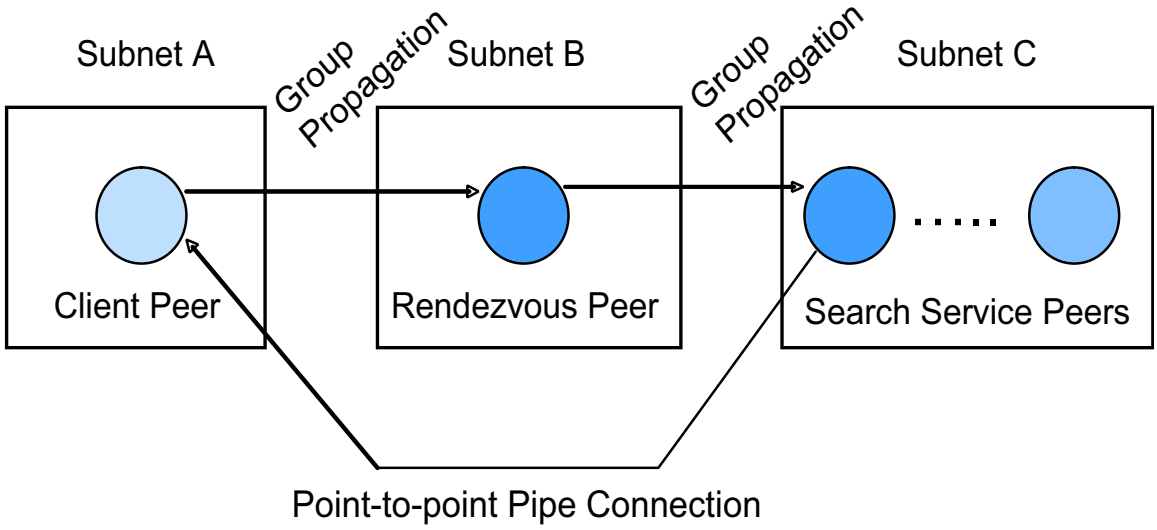

Figure 7.4. The Message Response Setup in P2P Networks

and Java Hotspot VM 1.4.2. The internal network bandwidth of the subnet C is 1 Gbps, and the switch for the outside connections has 100 Mbps bandwidth. The lowest bandwidth for our computer network links including the hop connections is 100 Mbps.

We use JXTA version 2.3 for the performance measurement. To evaluate the rendezvous peer effect on P2P connections, we select two different cases—one has two rendezvous peers whose architecture is the same as in Figure 7.4, and the other has one rendezvous peer whose architecture is missing subnet B from Figure 7.4. We follow the experimental measurement method from the JXTA bench project [85]. The query client peer sends a message for 50 times with 1 second interval. The message is similar to a regular query—an arbitrary keyword and an author name. The response times are recorded and we calculate the average of total and maximum response times. The maximum response time means the longest time among search service peer query results. The highest 10 % of the response times on each rally are cut off to reject abnormal network communications. We adopt this criterion from benchmark programs in [85]. Each query result used in this measurement is a dummy string set with 1,024 characters—2 KB.

Figure 7.5 shows the average message response time for 2, 4, and 8 group search service peers. In this performance benchmark, one machine manages only one JXTA peer. The

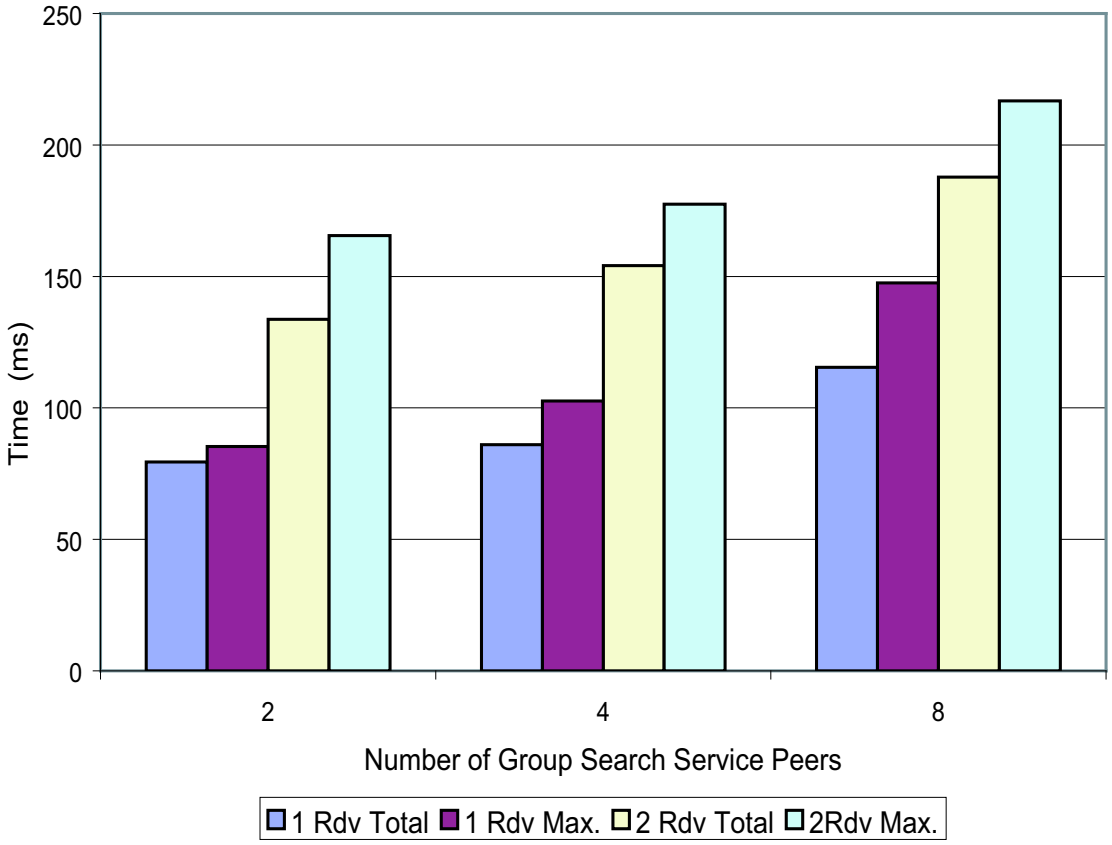

Figure 7.5. Average Message Response Time for a Query

communication overhead of an intermediate rendezvous peer is relatively stable in this graph. Figure 7.6 shows the average message response time from 1 to 16 group search service peers. This experiment only uses one rendezvous peer in search service peers. In 16 group peers case, each machine has two peers running. The 32 group peers case is not included in this graph, because it is unstable and many expected responses are missing. The reason will be discussed later in this section. Therefore, we draw this case with a different way but similar to that of JXTA bench. We plot each message response time as a point per each message sent. Figure 7.7 shows the benchmark results. JXTA bench also plotted JXTA propagate pipe performance results—throughputs—in Figure 7.8. They used 4 rendezvous peers with one seeding, 5 TCP edge peer, and 10 minutes load test run, but the detailed environment

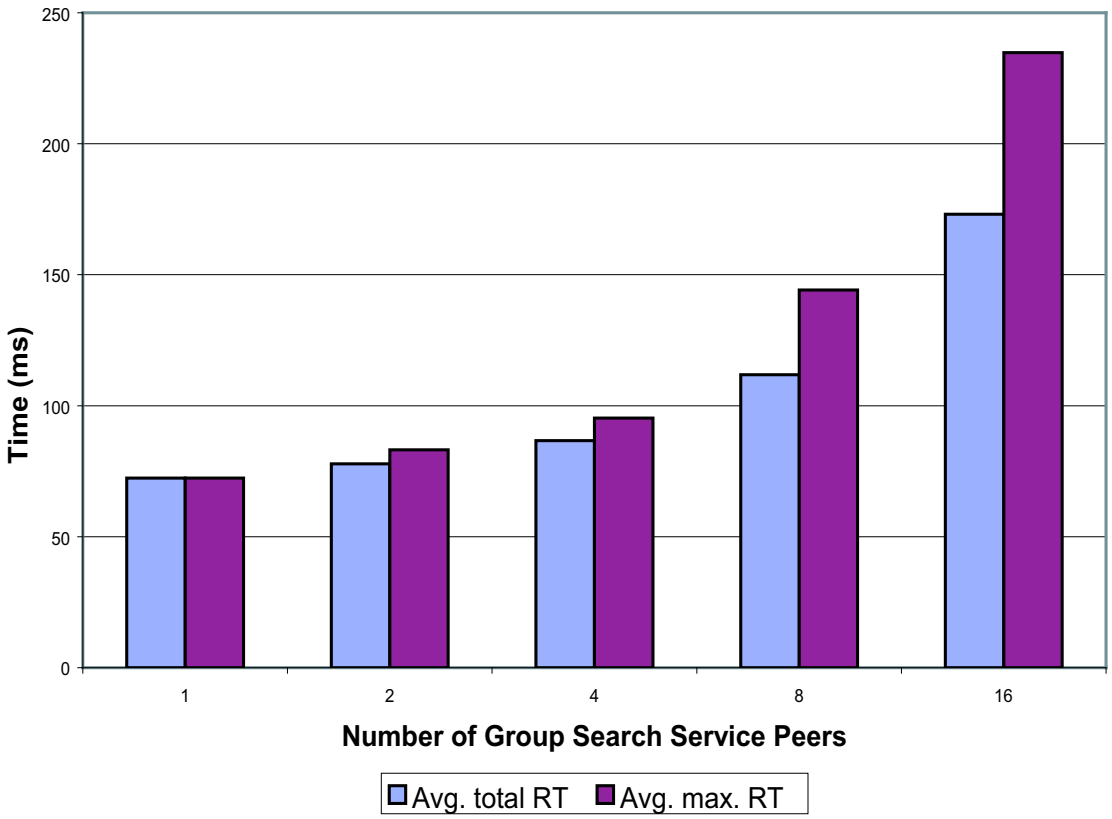

Figure 7.6. Average Response Time for a Query with Multiple Peers per Node Allowed

setup is not described on their Web pages. This graph also shows the increased response time for a while as in our experimental graph.

In a JXTA developer mailing list [86], Jeffrey Altman notes, "Multiple JXTA stacks on one machine should not be considered to be reliable when propagating via IP Multicast." We reinforce his statement through our experimental measurements. Actually the JXTA version 2.3 unlike JXTA version 1 implementation only supports multicasting for the pipe communication. If we turn off the multicast option in JXTA Configurator, the JXTA program just returns to shell prompt without execution due to lack of the implementation. As a result, further experimental tests may not extract a meaningful result with the expansion of the number of peers using the same number of machines.

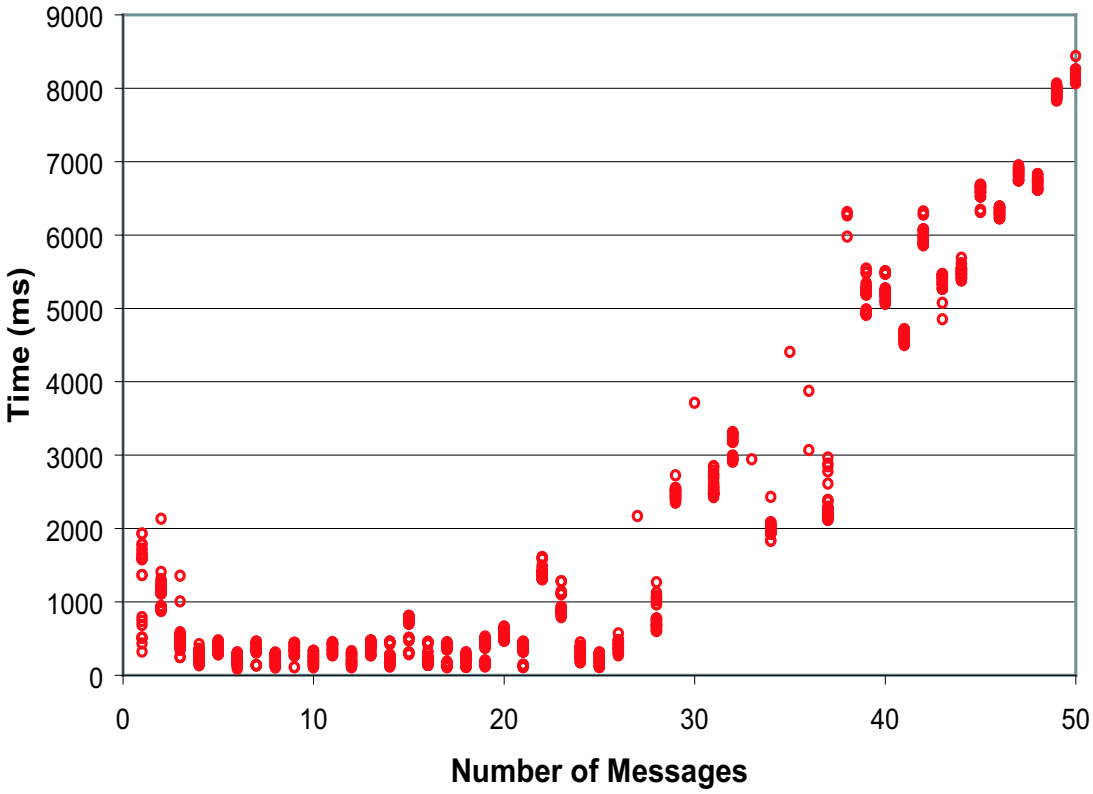

Figure 7.7. Message Response Time for 32 Group Peers

7.5 Discussion

Generally P2P systems can have larger scalability than event-based systems [87]. This is because P2P systems have an active discovery mechanism, whereas that of event-based systems is passive. The old Napster [8]—a P2P based music file sharing system—was a typical example for the huge scalability of a P2P system. Therefore, we can expect better scalability from the hybrid keyword search over P2P communications than that on distributed database based on message-oriented middleware (MOM).

We demonstrated our hybrid search across P2P federated databases, and produced some data from our experimental performance tests fundamentally based on a message flooding. Nonetheless it is hard to simulate how much scalable our system is from those data. Further

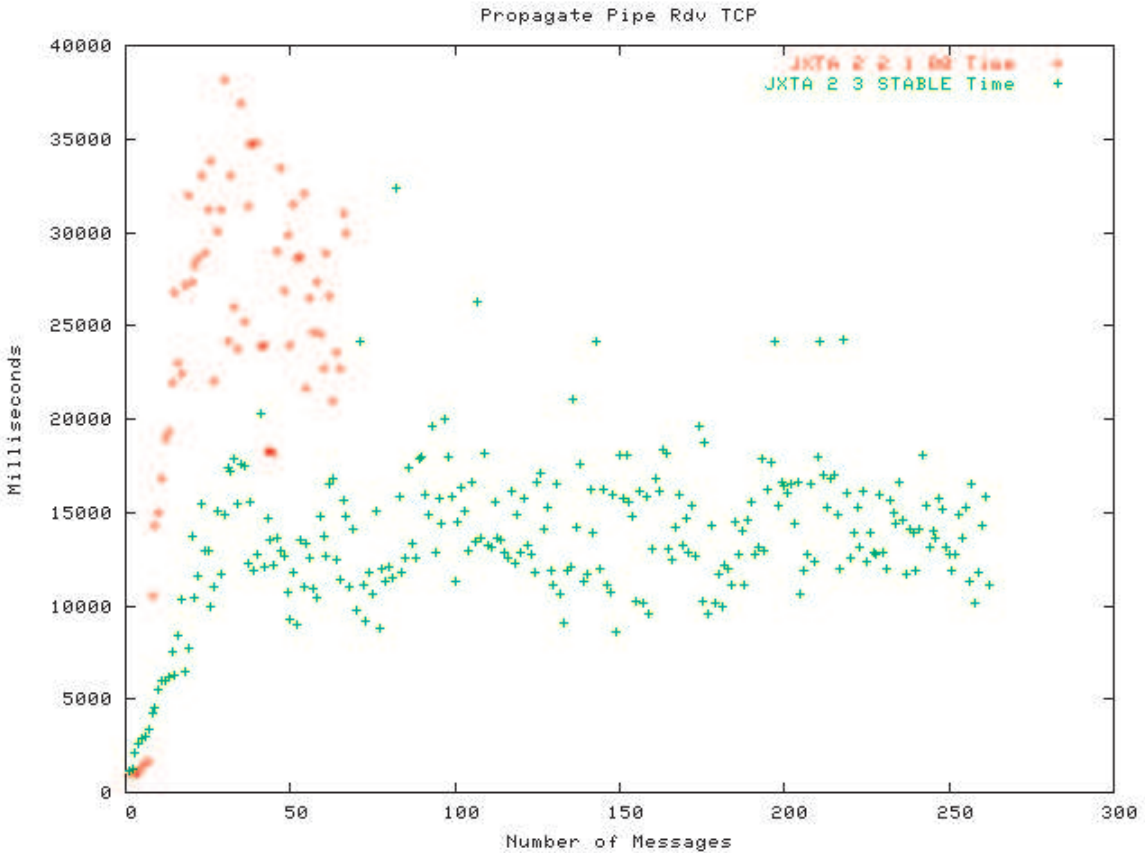

Figure 7.8. Response Time for Propagate Pipe Performance from JXTA bench

performance evaluations for the scalability based on a distributed algorithm are described in [88] in detail, but they concluded that the performance can not be predicted from simulation. Newer JXTA versions, after 2.0, introduced a Shared Resource Distributed Index (SRDI) mechanism over the JXTA rendezvous peer network. They showed that the SRDI paradigm improves communication performance compared to that of JXTA 1.0. The project JXTA is still underway and it improves toward a P2P communication tool for a huge scalability.

We will describe other related works that demonstrated keyword searches over distributed communication systems in the next chapter. Some of them claim their scalability could be reach to thousands nodes with a reasonable response time.

CHAPTER 8

RELATED WORKS

Since some file sharing applications on peer-to-peer overlay networks drew public attention a few years ago, much research on P2P networks has gone on. Many studies focus on distributed lookup in routing to reduce the unnecessary communications. The Distributed Hash Table (DHT) approaches such as Chord [52], CAN [53], Pastry [54], and Tapestry [55] are typical examples of such efforts. Reynolds and Vahdat [89] pointed out the lack of search capability in DHT based file storage applications. They focused on the reduction of the network traffic through the DHT oriented distributed search engine for "multiple-keyword" queries. The newer versions of JXTA [58] also introduced DHT in their index maintenance of rendezvous super-peers, but they use a hybrid approach stressing a "multiple random walks" based algorithm [90]. This is because DHT applies well only for the stable structure of peers.

ODISSEA (Open DIStributed Search Engine Architecture) [91] is an example of content search on P2P networks with distributed global index. Its distributing index of object names depends on a DHT structure—Pastry [54]. The study of Galanis and others [92] focused on query delivery to the right nodes through distributed directories (catalog). It used XPath as a query language, with the XML element names as indexes distributed through the Chord [52] DHT-based protocol.

Another interesting approach to look up peers is a technique based on *reputation*. In Gnutella-like P2P networks, a rating for the peers will help to choose the search target. Current reputation systems for P2P networks are described well in [93].

Gossiping is a way of information propagation for members of a group connected by a communication network, where "every person in the network knows a unique item of information and needs to communicate it to everyone else" [94]. Some researchers [95, 96] noticed the utility of gossiping in peer-to-peer networks, as the mechanism for a replicated

directory to locate the information. PlanetP [96] uses a gossiping algorithm to share the index and provides pure content search based on a peer-to-peer system. They claim that their gossiping paradigm is appropriate up to thousands of peers on a P2P overlay network. No metadata is assimilated in their work.

PeerDB [82] is a P2P distributed data sharing system that enables content-based search without shared schema. The query propagation depends on an agent in each peer. There is no specific policy, but Gnutella-like query propagation with a TTL value. The authors experimented on a cluster of computers.

Nakauchi and others [97] enlarged the category of keyword search on P2P networks with query expansion. The query expansion is based on a thesaurus keyword relational database with a distributed mechanism for updating. Hristidis and others [98] addressed the previous problems of keyword search in databases by exploiting a context rank for joining relevant tuples. This should be an effective strategy for keyword search on existing relational databases with reducing the lost part of the inherent meaning from database schema. However, our study focuses on keyword search over existing documents with additional metadata attachment to narrow the search category.

XRANK [99] focused on search over contents embedded in XML documents with semantic tags. This study emphasizes document-centric XML, and both semantic metadata and contents are included in the XML. However, this study does not include peer-to-peer communication.

CHAPTER 9

CONCLUSIONS

To discover and share heterogeneous resources on the Net has been a long term challenge since computer communication networks were popularized. There are two traditional approaches to organizing the data to be searched—structured data and unstructured data. Semistructured data based on structured data has recently emerged to answer the demands of the Internet environment.

Web search engines are rooted in searching against unstructured data. But organizing Internet resources as structured or semistructured data creates complexity, while unstructured data are hard to categorize without metadata. In this dissertation, we have described a hybrid search bridging the benefits from those two search approaches—simplicity plus some semantic value.

9.1 Contributions

The first contribution of this dissertation is the demonstration of a hybrid search—combining metadata search with a keyword search over unstructured context data. This search paradigm enables narrowing search category through metadata constraints over a legacy keyword-only search for unstructured data. We have demonstrated it with two architectures. One is utilizing an XML-enabled relational DBMS with nested subqueries to implement the combination of query results against unstructured documents and semistructured metadata. The other is based on a native XML database and a text search library. To associate metadata with unstructured documents, we have assigned the file name for the document as the key of the metadata. A hash table is used for a temporary storage of metadata query results and the keyword search maps to the table subsequently for joining.

The second contribution of this dissertation is a way to increase locality and integrate several dispersed resources through a data integration hub. A group of data integration hubs

may provide a global search over a distributed information system by using the cooperative network features built into a message broker, or by using an additional network layer—e.g. a P2P overlay network. One hub can be a search service node in a distributed database, or a peer in a P2P overlay network.

The third contribution of this dissertation is extension of the scalability of a native XML database and performance improvement for some queries compared to those on a single machine. They are possible through a scalable hybrid search on distributed databases. Those distributed architectures are mainly based on several or many computer network connections utilizing a message-oriented middleware or a peer-to-peer network framework. We have particularly focused on a total query result generated from a union of queries against data fragments in a computer cluster.

The final contribution of this dissertation is a generalization of our hybrid search architecture on potentially more scalable P2P overlay network. In this architecture queries over P2P overlay networks can be partial matches. This generalization may be a practical intermediate search on the Internet—providing semantic value through XML metadata that are much simpler than those of the Semantic Web.

9.2 Possibilities for Future Work

There are many interesting possibilities for the future work. Regarding querying against XML storages, our experience suggested that context based indexing over XML instances can improve the query performance. Adding such a feature on the Xindice XML database one expects better XML query performance. We did not designate any schema for the XML metadata query, but further studies for effects of approximate query against XML instances are desirable. That may enable more general and scalable inquiries among resources on P2P overlay networks.

Efficient routing and load balance through the metadata in the hybrid search on P2P communication networks have plenty of research possibilities. Though our experimental performance tests focused on message flooding due to the limitation of experimental environments, P2P communications gain a superior position when they utilize their embedded automated routing mechanism. Those potential experiments can be extended to comparisons of communication performances on several P2P overlay network frameworks.

Though P2P communication is designed for huge scalability, we restricted our performance measurements to a computer cluster. We may extend our evaluation of the communication tools to the case of worldwide distribution.

9.3 Publications

1. J. Kim, G. Fox, and S. Yoo. Data Integration Hub for a Hybrid Paper Search. Accepted for presentation in *9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES)*, To appear in Lecture Notes in Artificial Intelligence. September, 2005.

2. J. Kim and G. Fox. Scalable Hybrid Search on Distributed Databases. Accepted for presentation in *3rd International Workshop on Autonomic Distributed Data and Storage Systems Management (ADSM)* in conjunction with ICCS, To appear in Lecture Notes in Computer Science. May, 2005.

3. J. Kim and G. Fox. A Hybrid Keyword Search across Peer-to-Peer Federated Databases. In *Proceedings of 8th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, September, 2004.

4. J. Kim, O. Balsoy, M. Pierce, and G. Fox. Design of a Hybrid Search in the Online Knowledge Center. In *Proceedings of IASTED International Conference on Information and Knowledge Sharing*, November, 2002.

5. G. Aydin, H. Altay, M. S. Aktas, M. N. Aysan, G. Fox, C. Ikibas, J. Kim, A. Kaplan, A. E. Topcu, M. Pierce, B. Yildiz, and O. Balsoy. Online Knowledge Center Tools for Metadata Management. Technical report, DoD HPCMP Users Group Meeting, June, 2003.

6. O. Balsoy, M. S. Aktas, G. Aydin, M. N. Aysan, C. Ikibas, A. Kaplan, J. Kim, M. Pierce, A. Topcu, B. Yildiz, and G. Fox. The Online Knowledge Center: Building a Component Based Portal. In *Proceedings of the International Conference on Information and Knowledge Engineering*, June, 2002.

7. G. Fox, S. Ko, M. Pierce, O. Balsoy, J. Kim, S. Lee, K. Kim, S. Oh, X. Rao, M. Varank, H. Bulut, G. Gunduz, X. Qiu, S. Pallickara, A. Uyar, and C. Youn. Grid services for earthquake science. *Concurrency and Computation: Practice and Experience*, 14:371—393, May—June 2002.

REFERENCES

[1] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of International World-Wide Web Conference*, April 1998.

[2] World Wide Consortium (W3C). Semantic Web. World Wide Web. `http://www.w3.org/2001/sw/`.

[3] S. Decker, S. Melnik, F. Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, September–October 2000.

[4] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of the International Conference on Data Engineering(ICDE)*, February 2002.

[5] A. Hulgeri, G. Bhalotia, C. Nakhe, and S. Chakrabarti. Keyword Search in Databases. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, September 2001.

[6] D. Quass, A. Rajaraman, and J. Widom Y. Sagiv, J. Ullman. Querying Semistructured Heterogeneous Information. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases*, 1995.

[7] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. *Computer Networks*, 31(11–16):1155–1169, 1999.

[8] Napster. World Wide Web. `http://www.napster.com/`.

[9] SETI@Home. World Wide Web. `http://setiathome.ssl.berkeley.edu/`.

[10] Sun Microsystems. The JXTA project and Peer-to-Peer Technology. World Wide Web. `http://www.jxta.org/`.

[11] T. Bray, J. Paoli, C. Sperberg-McQueen, and E. Maler (Eds). Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web, October 2000. `http://www.w3.org/TR/REC-xml`.

[12] D. Raggett, A. Hors, and I. Jacobs (Eds). HTML 4.01 Specification. World Wide Web, December 1999. `http://www.w3.org/TR/html4/`.

[13] World Wide Consortium (W3C). XML Schema. World Wide Web, May 2001. `http://www.w3.org/XML/Schema`.

[14] O. Lassila and R. Swick (Eds). Resource Description Framework (RDF) Model and Syntax Specification. World Wide Web, February 1999. http://www.w3.org/TR/REC-rdf-syntax/.

[15] D. Brickley and R. V. Ghuha (Eds). RDF Vocabulary Description Language 1.0: RDF Schema. World Wide Web, December 2003. http://www.w3.org/TR/rdf-schema/.

[16] S. DeRose, E. Maler, and D. Orchard (Eds). XML Linking Language (XLink) 1.0. World Wide Web, December 2000. http://www.w3.org/TR/xlink/.

[17] S. DeRose, R. Daniel Jr., P. Grosso, E. Maler, J. Marsh, and N. Walsh (Eds). XML Pointer Language (XPointer). World Wide Web, August 2002. http://www.w3.org/TR/xptr/.

[18] J. Clark and S. DeRose (Eds). XML Path Language (XPath) 1.0. World Wide Web, November 1999. http://www.w3.org/TR/xpath.

[19] A. Berlgund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon (Eds). XML Path Language (XPath) 2.0. World Wide Web, November 2003. http://www.w3.org/TR/xpath20/.

[20] International Organization for Standardization. Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. ISO 8879–1986.

[21] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proceedings of the Workshop on the Web and Databases (WebDB)*, June 1999.

[22] International Organization for Standardization. Information Technology—Syntactic Metalanguage—Extended BNF, 1996. ISO/IEC 14977.

[23] World Wide Consortium (W3C). DAML+OIL Web Ontology Language. World Wide Web, December 2001. http://www.w3.org/Submission/2001/12/.

[24] I. Horrocks. DAML+OIL: a Reason-able Web Ontology Language. In *Proceedings of the International Conference on Extending Database Technology(EDBT)*, March 2002.

[25] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Pastel-Schneider, and L. Stein. OWL Web Ontology Language Reference. World Wide Web, December 2003. http://www.w3.org/TR/owl-ref/.

[26] T. Berners-Lee. Why RDF model is different from the XML model. World Wide Web, September 1998. http://www.w3.org/DesignIssues/RDF-XML.html.

[27] J. Clark (Eds). XSL Transformations (XSLT) Version 1.0. World Wide Web, November 1999. http://www.w3.org/TR/xslt.

[28] R. Bourret. XML and Databases. World Wide Web. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[29] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. Oracle8i—The XML Enabled Data Management System. In *Proceedings of the International Conference on Data Engineering(ICDE)*, February 2000.

[30] Oracle Corporation. *Oracle9i Application Developers Guide—XML*, June 2001.

[31] Oracle Corporation. *XML Database Developer's Guide—Oracle XML DB Release 2 (9.2)*, March 2002.

[32] Oracle Corporation. *Oracle XML DB Developer's Guide 10g Release 1 (10.1)*, December 2003.

[33] J. Cheng and J. Xu. XML and DB2. In *Proceedings of the International Conference on Data Engineering(ICDE)*, February 2000.

[34] M. Rys. State-of-the-Art XML Support in RDBMS: Microsoft SQL Servers XML Features. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, June 2001.

[35] A. Conrad. A Survey of Microsoft SQL Server 2000 XML Features. World Wide Web, March 2000. `http://msdn.microsoft.com/library/`.

[36] A. Deutsch, M. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. In *Proceedings of ACM SIGMOD Conference*, June 1999.

[37] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, September 1999.

[38] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proceedings of the International Conference on Data Engineering(ICDE)*, September 1995.

[39] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[40] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, September 1997.

[41] H. Schoning. Tamino—a DBMS Designed for XML. In *Proceedings of the International Conference on Data Engineering(ICDE)*, April 2001.

[42] Apache Software Foundation. Xindice. World Wide Web. `http://xml.apache.org/xindice/`.

[43] dbXML Group. dbXML. World Wide Web. `http://www.dbXML.org/`.

[44] Sleepycat Software. Berkeley DB XML. World Wide Web. `http://www.sleepycat.com/products/xml.shtml`.

[45] M. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In *Proceedings of USENIX Technical Conference*, June 1999.

[46] R. Bourret. XML Database Products. World Wide Web, 2003. `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`.

[47] The Free Network Project. World Wide Web. `http://freenet.sourceforge.net/`.

[48] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*. Springer, 2001.

[49] Napster protocol open specification. World Wide Web, April 2000. `http://opennap.sourceforge.net/napster.txt`.

[50] Gnutella. World Wide Web. `http://gnutella.wego.com/`.

[51] The Gnutella protocol specification v0.4. World Wide Web. `http://www.clip2.com/`.

[52] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peertopeer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM Conference*, August 2001.

[53] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM Conference*, August 2001.

[54] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms(Middleware)*, November 2001.

[55] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, U.C. Berkeley, April 2001.

[56] FastTrack. World Wide Web. `http://www.fasttrack.nu/`.

[57] JXTA v2.0 Protocols Specification. World Wid Web, March 2003. `http://spec.jxta.org/`.

[58] B. Travesat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. Hugly, E. Pouyoul, and B. Yeager. Project JXTA 2.0 Super-Peer Virtual Network. World Wide Web, May 2003.

[59] A. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[60] G. Banavar, T. Chandra, R. Strom, and D. Sturman. A Case for Message Oriented Middleware. In *Proceedings of the International Symposium on Distributed Computing(DISC)*, September 1999.

[61] Sun Microsystems. Java Message Service Specification Final Release 1.1. World Wide Web, April 2002. http://java.sun.com/products/jms/docs.html.

[62] S. Pallickara and G. C. Fox. NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In *Proceedings of International Middleware Conference*, June 2003.

[63] J. Kim, O. Balsoy, M. Pierce, and G. Fox. Design of a Hybrid Search in the Online Knowledge Center. In *Proceedings of the IASTED International Conference on Information and Knowledge Sharing*, November 2002.

[64] NPAC. Northeast parallel architectures center. World Wide Web, 2000. http://www.npac.syr.edu.

[65] G. Salton. *Automatic Text Processing: The Transformation Analysis and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[66] C. Faloutsos. Access Methods for Text. *ACM Computing Surveys*, March 1985.

[67] A. Maier and D. Simmen. DB2 Optimization in Support of Full Text Search. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, December 2001.

[68] J. Hamilton and T. Nayak. Microsoft SQL Server Full-Text Search. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, December 2001.

[69] P. Dixon. Basics of Oracle Text Retrieval. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, December 2001.

[70] Oracle Corporation. *Oracle Text—Reference*, March 2002.

[71] Apache Software Foundation. Jakarta Lucene. World Wide Web. http://jakarta.apache.org/lucene/.

[72] Oracle Corporation. Oracle9i Release 1 and XML:Leveraging the Oracle9i XMLType. World Wide Web, August 2001. http://www.oracle.com/technology/tech/xml/htdocs/XDBDemo1_Code.html.

[73] M. Ley. Computer Science Bibliography. World Wide Web. http://www.informatik.uni-trier.de/\~{}ley/db/.

[74] W. Hersh, C. Buckley, T. Leone, and D. Hickam. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th Annual ACM SIGIR Conference*, 1994.

[75] T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[76] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[77] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[78] F. Berman, G. Fox, and A. Hey, editors. *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[79] N. Alpdemir, A. Mukherjee, N. Paton, and P. Watson. Service-Based Distributed Querying on the Grid. In *Proceedings of International Conference on Service Oriented Computing (ICSOC)*, December 2003.

[80] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard (Eds.). Web Services Architecture: W3C Working Group Note 11 February 2004. World Wide Web, February 2004. `http://www.w3.org/TR/ws-arch/`.

[81] G. Fox, S. Pallickara, and S. Parastatidis. Towards Flexible Messaging for SOAP Based Services. In *Proceedings of International Conference for High Performance Computing and Communications(SC)*, November 2004.

[82] W. Ng, B. Ooi, K. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *Proceedings of the International Conference on Data Engineering(ICDE)*, March 2003.

[83] E. Halepovic and R. Deters. The Costs of Using JXTA. In *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*, September 2003.

[84] E. Halepovic and R. Deters. The JXTA Performance Model and Evaluation. *Future Generation Computer Systems*, March 2005.

[85] JXTA Bench Project. World Wide Web. `http://bench.jxta.org/`.

[86] JXTA Discussion Alias for Technical and Related Topics for Developers. World Wide Web. `http://www.jxta.org/servlets/SummarizeList?listName=dev`.

[87] A. Aberer and M. Hauswirth. Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems. A tutorial presentation in International Conference on Data Engineering (ICDE), February 2002.

[88] E. Halepovic, R. Deters, and B. Traversat. Performance Evaluation of JXTA Rendezvous. In *Proceedings of International Symposium on Distributed Objects and Applications (DOA)*, October 2004.

[89] R. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Proceedings of International Middleware Conference*, June 2003.

[90] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of ACM International Conference on Supercomputing(ICS)*, Jun 2002.

[91] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *Proceedings of the Workshop on the Web and Databases (WebDB)*, June 2003.

[92] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, September 2003.

[93] M. Gupta, P. Judge, and M. Ammar. A Reputation System for Peer-to-Peer Networks. In *Proceedings of International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2003.

[94] S. Hedetniemi, T. Hedetniemi, and A. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18:319–349, 1988.

[95] H. Johansen and D. Johansen. Improving Object Search using Hints, Gossip, and Supernodes. In *Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS)*, October 2002.

[96] F. Cuenca-Acuna, C. Peery, R. Martin, and T. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing(HPDC)*, June 2003.

[97] K. Nakauchi, Y. Ishikawa, H. Morikawa, and T. Aoyama. Peer-to-Peer Keyword Search Using Keword Relationship. In *Proceedings of International Workshop on Global and Peer-to-Peer Computing (GP2PC)*, May 2003.

[98] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, September 2003.

[99] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of ACM SIGMOD Conference*, June 2003.