WWW Search Systems Using SQL*TextRetrieval and Parallel Server for Structured and Unstructured Data[1]

Gang Cheng, Piotr Sokolowski[2] , Marek Podgorny and Geoffrey Fox
[gcheng,piotr,marek,gcf]@npac.syr.edu

Northeast Parallel Architectures Center at Syracuse University, NY 13244

Abstract

We describe our experience in developing Web Search Systems using Oracle's SQL*TextRetrieval. In the prototype system we store on-line books in the HTML and the HTML documents of a web site, SQL*TextRetrieval is used to index full text and other structured data in the 'web space' and to provide an efficient search engine for free-text search. The Web enables global access to and maximum information sharing with a hypertext-based text retrieval system. Using Oracle's Free Text Retrieval technology various search options are implemented, including basic word stemming, phrase, fuzzy, and soundex searching, as well as more advanced proximity search and concept search. For the concept search option, we have integrated a public domain "Roget Thesaurus" into our text search system to support synonym expansions. An advanced search mechanism to recursively refine search domain via the web is also described. The prototype system can be found at URL <*http://kayak.npac.syr.edu:1963/search/index.html*>. A full production system will be implemented on a multiprocessor parallel machine where parallel Oracle 7 with parallel server an query options are used.

1. Introduction

World Wide Web (WWW) has become a common place and a popular way for either corporate or Internet users to publish and exchange information over networks. Base components of WWW include: HTML - the markup language; HTTP - the networking transport protocol; HTTP server and web browser - the server and the client. WWW can be viewed as a giant distributed database (or data repository) where data entities are stored in HTML pages on web servers which are mostly non- or poorly-structured text documents and where logical relationships among web pages are represented by 'Hyperlinks' in HTML. The interactions between a web browser and (remote) web servers via HTTP can be seen as the (network-based) access interface/transactions between the database client and server, where browsing, i.e., clicking by following hyperlinks, is the simplest and most primitive search/query activity in the database such that no sophisticated search engine is required except for the file system on the web server. Depending on the client's browsing scope, the content of the distributed database can be as small as a single web site, or it can be as big as all the web servers on the whole Internet. With the daily growing volume in the WWW space, it becomes impossible to solely rely on 'browsing' to quickly and accurately locate/discovery relevant information on the WWW. One natural solution is to build web search systems. For the distributed WWW database, one can view such a web search system as a centralized meta-index database where text indexes of the distributed data repository are built and stored at a central place to speedup the 'browsing' process.

---

[1] Presented at Oracle Developers Conference ECO'96, Aprit 22-23 1996, McLean Virginia.
[2] On internship from EFP (The Franco-Polish School of New Information and Communication Technologies), Poznan Poland

There have been many approaches to build such a web search system. For example, Yahoo [11] used a 'Yellow Page' paradigm to catalog web servers by some pre-defined criteria. While many WWW indexing services like Yahoo adopt such approaches close to WWW technology, in this paper we are mostly interested in employing a well-established, mature database and information retrieval technology to build the web search systems. This approach has also been used in the current main-stream web search systems, including Alta Vista [2], OpenText [5], Lycos [4], and InfoSeek [3].

This paper is organized as follows: Section 2 discusses system configuration and major components found in a typical web search system. Oracle's full-text indexing and search system "SQL*TextRetrieval" is described in Section 3. We describe our experience in developing a prototype web search system using SQL*TextRetrieval in Section 4. The last section outlines our current and planned research and development in this area.

2. Components of a WWW search system

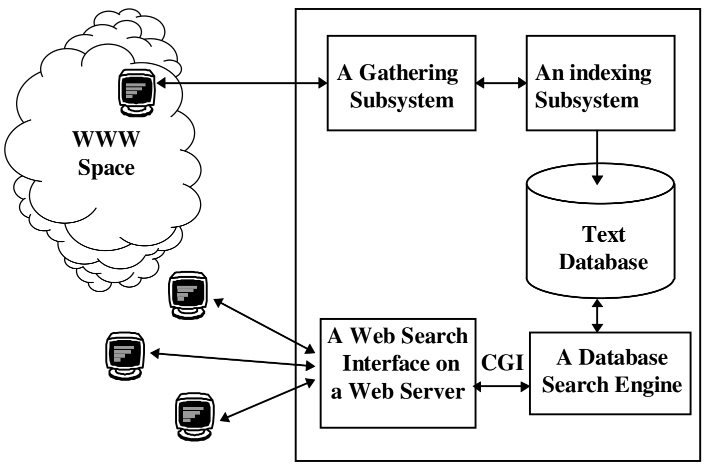

Figure 1. Components of A Web Search System

A web search system is a facility to locate/discover relevant information contained in the WWW URLs. The search domain can either be the whole WWW information space (including httpd, ftp, gopher, wais and USENET news servers) on the Internet or HTML documents on a single web site.

As shown in Figure 1, a web search system generally has four major components: a gathering subsystem, an indexing subsystem, a search engine and a Web-based search interface.

2.1 Web Gathering Subsystem

A Web Gathering subsystem's major function is to gather web pages/files from either remote or local WWW servers. It may involve either the whole Internet 'web space' which includes information on httpd, ftp, gopher, wais and USENET news servers or HTML documents on a single Web server. This gathering task is usually carried out automatically by a relatively small program sometimes called 'web robot' or 'web spider' or 'web agent' whose functionality can be briefly described as follows:

(1) Starting from a single URL, it requests a web page from the remote web site. Upon receiving the full page, it parses and adds all the URLs in this page to a queue for further gathering.
(2) For the web page from (1), according to the indexing rules predefined in the indexing subsystem and the search engine, it parses the page and filters only the information needed by indexing subsystem.
(3) After (2) is done, it gets a new URL from the queue, checks if the new URL is already indexed and proceeds with (1) again to repeat the gathering process.

In summary, a web robot continuously requests files from remote web sites and parses and filters each file for indexing and further gathering.

It is up to the gathering subsystem to decide where from, what and when the information will be cached/indexed/updated into the text database for web search. It determines the information content and volume, and, to a degree, the performance of a web search system. It can also have significant impact on the accuracy of a web search system. Different web systems may use different rules and approaches. Major issues and design parameters for the gathering subsystem are:

(1) Where from to gather
- all or only HTTPD,FTP,GOPHER,WAIS,NEWS server
- for a single server, all or first N levels of URLs in the 'hyperlinks tree'
- web pages in single or multiple text formats of HTML, Plain Text, Latex etc.

(2) What to index
- all words or parts of 'significant' keywords in a web page, i.e., full-text or keywords only
- attributes of a file, such as last update date, title, subtitles, size, outline etc.
- controlled by size/lines, e.g. first 10 lines of a text file .

(3) When to gather/index/update
- the same as or a separate indexing database from the current one used by web search interface choice
- real time (i.e. online indexing) or batch during nights.

(4) Performance of a gather -- number of files that can be gathered/indexed per day/hour

2.2 Indexing Subsystem

The indexing subsystem defines the how contents gathered from remote web sites is internally stored and managed in a text database to be able to efficiently and effectively support searching. It depends on the logical data structure of text entities and the physical organization/layout of textual indexes in the database. The indexing subsystem is tightly coupled with the search engine as its sole purpose is to speed up free text search/query process conducted by the search engine. Common approach is to build an *'inverted index'* for all keywords in a document.

The major decisions to be made for the indexing subsystems are:

(1) Compression scheme used to store the text and their indexes.
(2) Do we keep both original text and indexes in the database, or just keep indexes? In order to support certain advanced search capability (to be discussed in Section 2.4) such as phrase search and proximity search, the original text must be stored together with their indexes. This will increase the space consumption of the text database.
(3) Index modes: real-time indexing, batch indexing, or incremental indexing. Real-time indexing allows updates, inserts or deletes of text in/from the existing text database at any time, which will automatically modify the associated indexes, without affecting the simultaneous text retrieval on the index database. Batch indexing only updates index in a batch mode after a bulk of texts has been modified/loaded. Incremental indexing allows indexing process to be done incrementally as only the documents which have changed since the last indexing run are indexed. Thus, adding the new text will not affect the existing indexes in the database.
(4) Case sensitivity, symbols or numbers. Support of case sensitivity and symbols in the index will increase the space requirements.

Performance of a web search system will largely depend on how good its index scheme is. A good index scheme should have moderate storage requirements while significantly speeding up the query processing.

2.3 Search Engine Subsystem

The search engine subsystem accepts queries from a Web-based search interface and schedules, partitions and executes them against the indexed database to locate URLs and their associated attributes that satisfy the query criteria. It also deals with the I/O, caching, assembling, sorting and performance related system activity. Based on what content is indexed and on the scheme used in the indexing subsystem, it implements/supports different search algorithms to provide various search capabilities of a text retrieval system. Together with the previous two subsystems, the search engine determines what basic and advanced search capability a system can support and it is critical for the overall system performance.

Most Web search systems currently online are keyword-based search systems. The major search functions include:

Basic search capability:

(1) keywords with logical operators and their combinations, including 'and', 'or', 'not'.
(2) regular expression of keywords, including single/multiple wildcard matching.
(3) keyword expansion: stemming, fuzzy, soundex, expansions based on a thesaurus (such as synonyms).
(4) ranking of query results
(5) case sensitive/insensitive

Advanced search capability:
(1) summarizing - summarize a document
(2) similarity search - search 'similar' documents to a particular document
(3) phrase search
(4) proximity search - specify words distance between keywords

2.4 A Web-based Search Interface

Any information retrieval system has the two core subsystems discussed above - "indexing" and "search engine". They represent base technology developed in information processing literature and are used directly in the Web search systems. What is unique and different in a web search system, compared to a traditional text retrieval system, lies in the 'web gathering' and 'web search interface' subsystems. These two subsystems are also the major focus of our development work we have undertaken, as we rely on Oracle's RDBMS and SQL*TextRetrieval technology to build the 'search engine' and 'indexing' subsystems.

A conventional text retrieval system usually has its own search interface and also allows developers to build new search interfaces to access the text database. For example, SQL*TextRetrieval provides both form-based and C-based APIs. Most text retrieval systems use client-server model as its fundamental model to build access interface to end-users. Therefore, the client-server model used in WWW makes it seamless and natural to integrate a text retrieval system into a web search system, where a web client becomes a client of a text retrieval system. Due to the stateless (or sessionless) feature inherent to the web client-server model, new techniques are required in a web search interface which are quite different from those used in a traditional search interface design.

The major issues in developing a web search interface are:

(1) Integration of a web server and the backend search engine, i.e., interaction between a web server and the search engine, to efficiently facilitate the single transaction path:
query requests ->query processing ->query results. While interaction between a web client and web server is handled by the common http protocol, the mechanism of the interaction between a web server and a search engine may vary in different web search systems. Because a web search system is designed to handle large number of requests (which are, essentially, OLTP read-only transactions)

simultaneously, and a heavy load is assumed on the server side, high performance server technology is usually required to support scalable CPU and I/O processing. Another important consideration is how the system handles simultaneous search and database updates/indexing in real time.

Most current web search systems use some very limited 'parallel processing' techniques and replication technology to handle performance scalability issues. For instance, a common approach popularly used in Lycos, InfoSeek, OpenText etc. is to use more than one workstations, each running a web server and a local database. The local database on each machine is identical across all the machines, replicated from the machine dedicated to gather/index new web pages. The replication (thus the updates) of the local database usually is done once a day during night (some once a week/month). As far as we can see, this is a far cry from the modern database replication techniques.

(2) Sessionless web connections towards session-oriented database transactions on the web search interface. In many situations, session-oriented memorizing capability is required for the web search activity. For example, one query may generate a copious output that is too costly to send back all the way down to the display of the client's browser due to networking bandwidth and cache/memory constraints of the client machine. One possible approach is to store all the results on the server and provide navigation buttons (next, previous etc.) to guide the user in going through the query results in subsets. Because each user may issue a different query and the navigation requires more than one web transactions, the web search system must 'remember' the first query's results for each concurrent client. This means that a user's web session includes the first query and successive navigation transactions. In the early days of the web search systems, the only solution used to solve this problem was to use a 'Max hits' button for the first query so that the system restricted the maximum number of query results to be returned to the client. Currently, the session-oriented transaction are becoming a common practice in most web search systems.

(3) Query refining. This capability implies that after a query is issued whose search domain may be the whole indexed database, the new query uses the results returned in the previous query as its search domain. In that way, one can keep refining the search and finally find/narrow down to the required search targets.

(4) Support of the 'natural language like query'. Most web search systems only support keywords and their regular expressions to form a query. A good web search system should also allow users to type in queries using natural language. This function requires additional processing and analyzing of the query input from the search interface.

3. Oracle SQL*TextRetrieval

Traditional RDBMS systems are developed to deal with well-structured information entities such as business data. Information represented in WWW space is usually in the form of free-format with little or no pre-defined structure such as HTML pages, program source code and plain-ASCII articles. Because HTML is a very loosely defined markup language, a HTML page can be loosely or even ill formatted. For example, not every page has a title (as enclosed by <title> and </title> tag in HTML), although it is obligatory and acceptable by any web browser. Although documents in free-text form can be stored in tables of a RDBMS server, the search for the content of the documents without predefined search terms (words, phrases, or concepts) requires a serial scan of the table and an exact pattern matching on the text column. Performance of such a search is not acceptable. The standard indexing techniques used in Oracle RDBMS cannot help in this case, as the full-text documents are stored in a single or multiple table columns of character type. New indexing facility, together with its text search utility, must be developed to solve the performance issue in free text searching. SQL*TextRetrieval is one of such solutions developed by Oracle Corp. SQL*TR indexes both text that is contained in the multimedia documents and in the textual description of the data. It allows users to perform free text search that returns a set of documents or database rows which match the selection criteria. Because SQL*TR is a software layer built on top of the core Oracle7 RDBMS engine, it uses SQL queries as the standard method and query language for retrieving both textual information and structured data. It can take full advantage of the Oracle7 server features such as concurrency control, recovery, data integrity, security, backup and server management. Users can combine searching of structured information, (such as title, author, size, date of a web page) with free text searching. Among many utilities provided in the SQL*TR package [6] to build a fully functional yet traditional information retrieval system, the indexing facility and text retrieval API (in C) are most relevant to building a WWW search system. We briefly outline the major parts and functionality of the two components in this section. Readers are refered to [6][7] for details of SQL*TR.

*3.1 Text Indexing in SQL*TextRetrieval*

SQL*TR uses a fully inverted index method. It is a proprietary bit-mapped location index rather than a conventional pointer system. The index is highly compressed and it can be cached in memory: it can achieve as little as 4:1 ratio of the original text to their text indexes.
As fewer disk I/O operations are required, the compressed index improves the retrieval speed.

As illustrated in Figure 2, the text index in SQL*TR consists of two tables: a word list and a location list. There are two indexing modes in SQL*TR to control what is to be indexed: the STOP mode and the PASS mode. In the STOP mode, all words are indexed except those that have previously been entered in the STOP wordlist, (typically 'and', 'or', 'the', etc.). In the PASS mode, the words to be indexed are put in the wordlist before indexing occurs and are flagged as pass words. During indexing, all other words in the documents are ignored. This provides a useful means at server side (as compared to at web gathering part) to filter out common words in a WWW text database to further reduce space consumption.

By default, lexical delimiters in SQL*TR that indicate the boundaries of words and numbers are all non-alphanumeric characters for words and all non-numeric characters for numbers with the exception of the decimal point or equivalent radix character. A user can exclude specified characters from the list of delimiters for a particular text table. This proves to be another very useful control mechanism in the indexing subsystem of a web search system. For example, we can define the starting and ending HTML tag symbols ('<' and '>') as two delimiters such that all text enclosed by them is not indexed. This effectively prevents the HTML tags in a web page from being indexed.

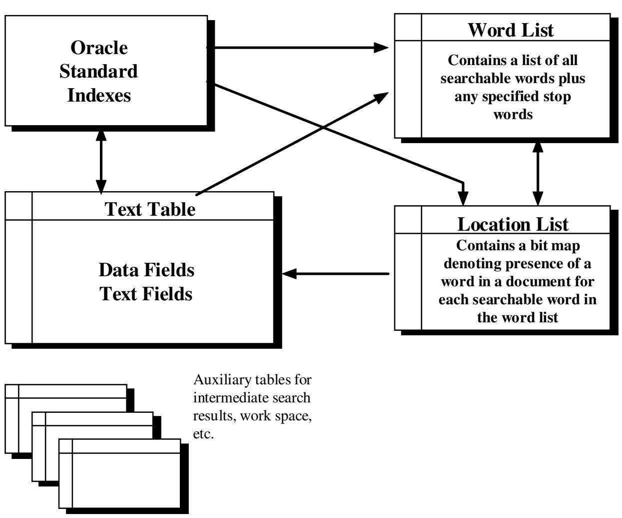

Figure 2. Oracle SQL*TR tables

The scope of an indexing run can be global (batch) or incremental, both at runtime while the searching is running. The Oracle RDBMS engine handles the concurrency control to support simultaneous text indexing and text retrieving. Updated text can be re-indexed either in real time or through a later run of incremental indexing. The advantage of real-time indexing to a web search system is that the updated text can be queried immediately. In real-time indexing, the text index is updated in the same DBMS transaction in which the text is added, changed, or deleted. All these features are required by an on-line web search system in which a single system supports simultaneous gathering/indexing and retrieving as WWW is a dynamic world - web pages are created, updated, removed from time to time. A web search system must reflect such a dynamic change within a reasonable time scale.

SQL*TR also provides a utility for loading and indexing large amount of data. The data can be a combination of structured data and free-format text. Data can be appended to tables so initial loads can be performed in several runs or large additional batches can be added while application are running.

3.2 Text Retrieval in SQL*TextRetrieval

SQL*TR's underlying retrieval language is based upon the extended SQL. This is accomplished by a comprehensive yet simple set of additions to SQL that enable searches to be carried out jointly on structured data and on the content of documents. Major syntactic extensions are the 'CONTAINS' clause (similar to 'LIKE') and a set of built-in functions to express different keyword search conditions. The language goes beyond mere search for keywords. A query can be made up of words, phrases, terms, and structured data operations. Standard Boolean of operations using AND, OR, and NOT can be used to build the queries.

Text tables in SQL*TR are basic Oracle tables with extended text columns where a text column holds parts of or the whole document in a free-text form. There are several text column types with various maximum sizes ranging from 255 bytes to unlimited size. Every text table must have a numeric column that holds a unique key known as textkey to locate specific rows in the table. Each row of the text table can be used to store free text of a document as well as any other structured information associated with the text (by using separate columns).

For a web search system, documents gathered from remote web sites are first loaded in a text table and later used to build their index database. After the indexing is finished, the original text data can be discarded, or it can be kept in the text table for some more advanced search purposes, to be discussed in Section 4. By its nature, unstructured text documents use large amount of storage. To save space, SQL*TR optionally compresses text stored in text tables. Considering the trade-off between the amount of space saved by compression and the time taken to decompress the text, there are two compression modes with space savings ranging from 40-60% to 60-70%.

Besides the SQL query language, SQL*TR provides two APIs for developers to build retrieval interface: Form API and Pro*C API. Ideally for a web search system, a PL/SQL based API of SQL*TR would be the best gateway to interface the text search engine built on SQL*TR (which is run on a Oracle7 RDBMS server) with the web search interface accessed on a HTTPD server, as demonstrated in the Web-Oracle-Web (WOW) package [10] and the Oracle WebSystem [8]. Such an interface is not supported at present. A set of Common Gateway Interface (CGI) programs written in the Pro*C API is currently the only solution to link a web server and an Oracle text server.

4. A prototype WWW search system

Our approach to build a web search systems was to develop a high-performance and intelligent web gathering subsystem and to build a flexible and customizable HTML form-based web search interface while employing Oracle RDBMS technology for the search engine and server management, and the SQL*TextRetrieval for the full-text indexing engine. Because the SQL*TR package was originally not developed for WWW information retrieval, much work has also been devoted to interface the gathering subsystem with the SQL*TR, and to integrate the web search interface with the Oracle7 RDBMS server.

Figure 3 is the system diagram of the web gathering subsystem developed for our web search system. It consists of three major components: a *loader*, a *checker* and one or more *gatherers*, each of which is an individual process. The fundamental philosophy behind this design is that the gathering program must be both high performance (the potential performance bottleneck of the limited networking bandwidth over

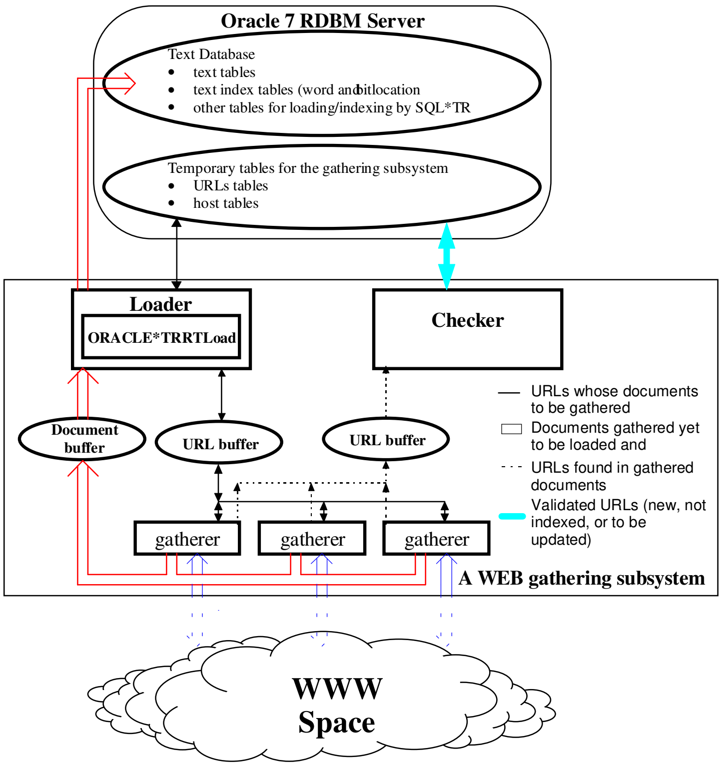

Figure 3. RDBMS based Web robot system

the wide area network should not become a bottleneck for the gathering process; the full advantage of an RDBMS server in concurrency control and caching should be utilized to build such a RDBMS-based web agent) and intelligent (it should detect the URLs that have different symbolic names but point to the same document, the web robot convention [1] should be followed, as well as it is extensible to gather domain/topic specific information).

During the gathering process, the *loader* and the *checker* are connected to the RDBMS server where tables consisting of the text and index database are stored. A set of temporary tables are also created to store URLs and hosts used to identify URLs that are new (to be gathered), pending, being processed or to be updated in the gathering processing by the *gatherers* and the *checker*. Concurrency control of accessing or updating these tables by the *checker* and the *loader* is managed by the RDBMS server. To sustain the maximum networking bandwidth, more than one *gatherer* can be started to interact with web servers simultaneously via HTTP to get bulk of documents in HTML. Each light-weighted *gatherer* parses the document to identify URLs to be considered for further gathering and a couple of attributes associated with the document, including URL, size, date, title, number of URLs and inline images, outline (first 256 characters) etc.

Major functions of the *loader* include:

(1) fetch new URLs to be gathered/updated from the Oracle server and pass them to *gatherers* upon request.
(2) invoke SQL*TR *'rtload'* process to load/update and/or index incrementally documents when they become available (in batch).

The *checker* is responsible for checking URLs:

(1) validate URLs to be gathered according to several filtering conditions:
- the disallowed URLs specified in the *'robot.txt'* on a host
- URLs with all the attributes (size, title, outline etc.) are identical yet may have different hostnames (alias) or paths (symbolic links)
- URLs found to be non-existed/removed in the previous visits of the *gatherer* (dead links)
- URLs being processed by the *loader* or one of the *gatherers*

(2) filter out URLs not to be gathered according to some predefined conditions, such as URLs for server statistics.
(3) filter in only URLs to be gathered according to predefined conditions, such as the online HTML books (*http://www.infomall.org/npac/pcw/**), NPAC web server (*http://www.npac.syr.edu/**) or any WWW server (*http://*,ftp://** etc).

Currently the information used by the *checker* to do the filtering is only the URL itself and the URL/host tables in the database. To make the gathering subsystem more intelligent, to be able to gather/index domain-specific information and such, the architecture design of our system is developed to be extensible/open. For example, we plan to add another module 'filter' between the *gatherer* and the *loader* which filters documents (not URLs) based on the content of the documents.

To enable the 'phrase search' and 'proximity search' in SQL*TR, the text of indexed documents need to be kept in the text table.

Figure 4 is the main search interface of the prototyping system where one of the indexed WWW database is the NPAC official web server. Other databases currently indexed include three online HTML books in parallel computing.

Directly built on top of the text retrieval in SQL*TR, this HTML form-based interface allows users to have the following search options:

(1) Logical operations of keywords - AND/OR ('all'/'any'), NOT ('contain'/'not contain')
(2) Word expansions
- word stemming - expanding a keyword based on its stem and indexed words having the same stem (wildcard matching on either end);
- fuzzy or soundex - words similarly spelled or phonetically similar;
- synonyms - same or nearly same meaning as other words predefined in the same synonym ring of a thesaurus. We converted a public domain "Roget Thesaurus" [9] into our database which has about 6000 synonym rings;
- search keywords to be treated as case sensitive/insensitive

(3) Phrase search - exact matching of a phrase;
(4) Proximity search - two words with relative location to each other.

The interface also provides access to a gateway, implemented using the WOW

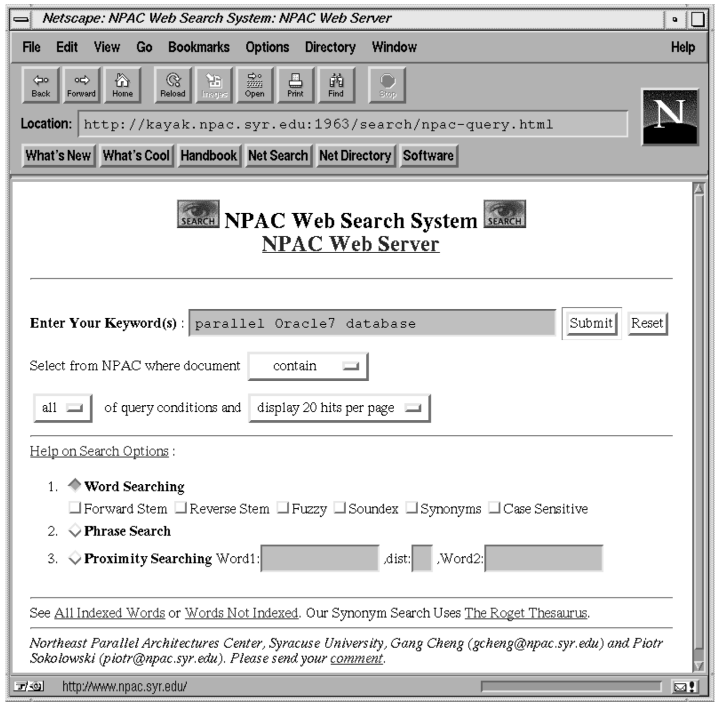

Figure 4: Main search interface of the NPAC web search system

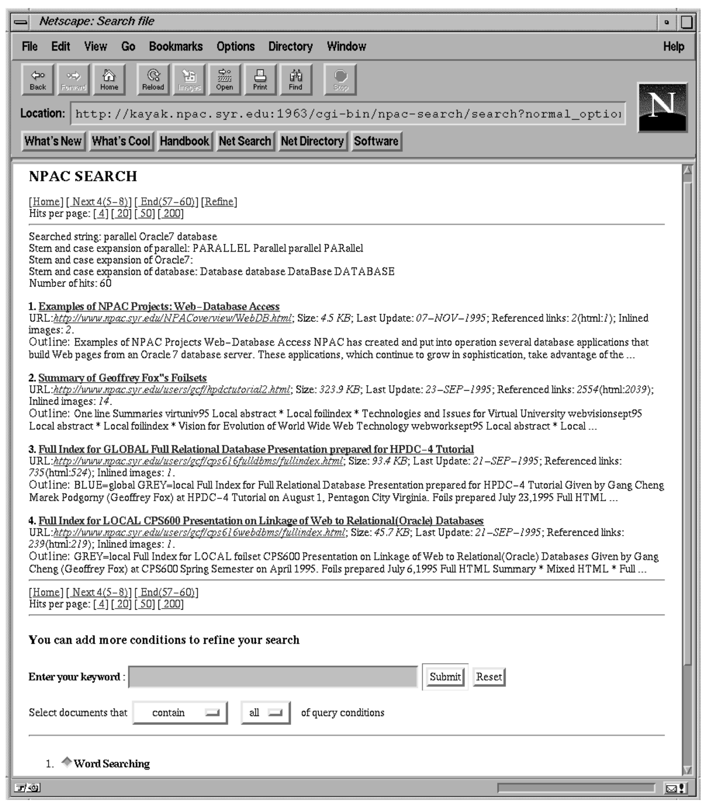

Figure 5: Display of search results and an interface of the navigation and refining

package, used to view all indexed words and stoplist words currently in the database, as well as a link to the source "Roget Thesaurus". This provides a simple server management layer.

As shown in Figure 5, search results are returned to the client's web browser one subset per page. Included in each result page are:

(1) A set of navigation buttons ('next', 'previous', 'top' and 'end') to allow the user to browse through all the query results page by page. SQL*TR's Stored Query Expressions (SQE), together with a sequence, is used to first store the query and its hitlist (i.e., identification of matching documents) and later restore them when these buttons are activated to request additional query results in successive web client-server connections. The use of SQE prevents the initial query from re-running. Note that the hitlist stored by SQE is in a highly compressed bitmap form so that the temporary space consumed should not be an issue on the server.

(2) An message area indicating search keywords and their expansions, and total number of matching documents found.

(3) The line items for matching documents that contain the search keywords, each of which is represented by a set of attributes associated with the document. The title of each document is hyperlinked back to the original URL for quick browsing of the full document source, while the URL attribute links to another query to pull out the document text from the database and make the search keywords to be highlighted in the display.

(4) A similar search interface to allow the user to refine the previous query so that the search domain of a new query will be the output of the previous query and not the whole database. This is again implemented by SQL*TR's SQE.

By using word expansions, we attempt to broaden the search domain to improve search accuracy limited by keywords search, while using the 'query refining' we support gradual narrowing of the search domain which may be over-extended by the word expansions.

5. Conclusions

As a conclusion of our current work, we point out that although SQL*TR provides us with an excellent free-text retrieval facility and testbed to rapidly prototype web search systems and investigate major issues in WWW-based information system with both well-structured and non-structured entities, we found a number of performance problems in the current SQL*TR package. For example, it appears that indexing information is not used in the 'phrase search' and 'proximity search' so they suffer very slow query response. We recently started to work on the new Oracle TextServer3 with ConText in our on-going web search development, and we hope that the new software will offer significant improvement in functionality and performance.

6. Current and planned work at NPAC

Northeast Parallel Architecture Center (NPAC) at Syracuse University has been engaged in research and development of advanced information systems, in particular enterprise information systems using advanced parallel processing and WWW technologies. Our current development work is focused on the two major areas:

(1) Investigate major development and performance issues in

- integrating conventional RDBMS technology in WWW-based systems,
- building production-level information systems using RDBMS and Web technologies

Our current technical strategy is to use Oracle 7 RDBMS server to build mail-based database, including USENET newsgroups, mailing lists and personal mailboxes, and use the system as a major vehicle or testbed to conduct/experiment major issues in Web and RDBMS integration. A production system of a searchable USENET newsgroups and mailing lists archive can be found at URL <*http://asknpac.npac.syr.edu*>.

(2) Web search systems using RDBMS and full-text indexing technology. As described in this paper, our approach is to use Oracle 7 RDBMS server as search engine and Oracle's Context server as full-text indexing engine to build backend web search engine, while developing our own web agents and web search interfaces. Supported by Oracle Corp., we are also conducting research and development on building large-scale web search and information systems on parallel machines. For example, we are currently using parallel Oracle7 with parallel server and query options to build WWW-based systems on a IBM Scalable POWERparallel (IBM SP2) system (*http://merlin9.npac.syr.edu:1963/*).

7. References

[1] A Standard for Robot Exclusion, *http://info.webcrawler.com/mak/projects/robots/norobots.html*.

[2] Alta Vista, *http://altavista.digital.com/*.

[3] InfoSeek, *http://guide.infoseek.com/*.

[4] Lycos, *http://www.lycos.com/*.

[5] Open Text, *http://www.opentext.com/omw/f-omw.html*.

[6] Oracle Corp. SQL*TextRetrieval Administrator's Guide V2.0 July 1992.

[7] Oracle Corp. SQL*TextRetrieval Developer's Guide V2.0 July 1992.

[8] Oracle WebSystem, *http://www.oracle.com/mainEvent/webSystem/webSystemOverview.html*.

[9] Roget Thesaurus, *http://www2.thesaurus.com/thesaurus/roget/*.

[10] Web-Oracle-Web, *http://dozer.us.oracle.com:8080/sdk10/wow/*.

[11] Yahoo, *http://www.yahoo.com/*.