Web Search System Based On Oracle7 and Oracle ConText Option

Gang Cheng, Yuping Zhu
[gcheng, yuzhu]@npac.syr.edu

Induction

With Internet's growth, World Wide Web becomes more and more popular. All over the world share infromation, people get what they want from any remote server using ftp, http and so on. But Web Space grows dramaticlly, how can people get what they want from so many infromation quickly , exactly and efficiently? Web Search System is the best solution so far. People just provide some hints, (e.g keyword, author, tiltle and so on) then they can get what they need.

Our Web Search System provides an efficient way to search HTML documents of one or several Web Sites. It combines Oracle 7 , Oracle ConText Option and Web Interface. We get all HTML documents from specific Web Sites using Web Robot, load them into Oracle, index full text and other attributes, provide Web Interface to search from any browser.

Components for Web Search System

From Macro's point of view, our Web Search System consists of three parts: Robot, Database and Web Interface.

1. Robot

Robot gathers documents from specific Web Site. The gathering range can be from a particular directory on certain Web Site to several Web Sites, even whole Web Space. It consists of four parts, as shown in Figure 1, each part is a seperate process. They are implemented using C/ Pro*C. They communicate each other using semaphore and message.

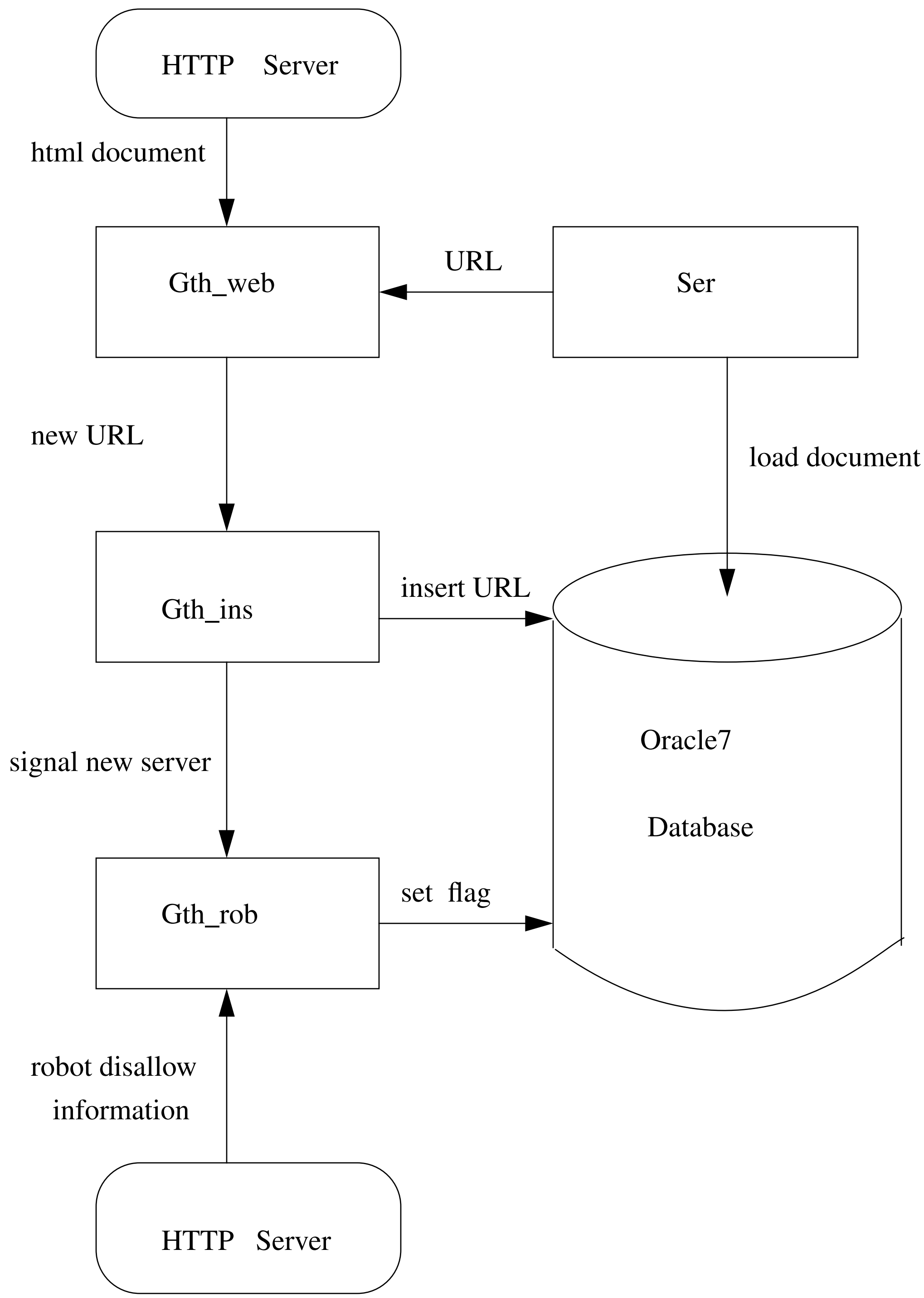

Figure 1. Web Robot System

1.1 gth_web

This part is written in C. It gets URL needed to be gathered from the process named ser, then creates socket to gather HTML document from specific HTTP Server, analyses the document, gets all information from it, (e.g all links, document size, last update time, total reference, total images and total forms), and format the document, (This is to guarantee loading the document into database correctly, e.g, discarding null character, and inserting newline character if there is no newline in whole document). After that, it appends all these information and whole document to a temparory file in order to load into database later, and sends new URLs gotten from the document to inspection process named gth_ins by message communication.

1.2 gth_ins

This part is written in Pro*C. It gets new URL from gahter part gth_web by message communication, checks it if the URL is valid, (e.g wether it belongs to the Web Space where you want to gather, wether it already exists in database). If URL is valid, inserts it into database in order to gather later. Otherwise discards it.

If it encounters new Web Server, it sends signal to robot_disallowed_check part gth_rob.

1.3 ser

This part is written in Pro*C. It manages semaphore and message communication, gets URL needed to be gathered from database, sends it to gather part gth_web , receives the infromation of gathering from gather part gth_web by message communication. It also loads documents into database.

1.4 gth_rob

This part is written in Pro*C. Its task is simple, just checks new Web Server if it allows robot to access, and sets relative flag.

2. Database

The Database part can be divided into three parts: storage, index and search.

2.1 Storage

We use relational database Oralce7 to store URLs, documents and relative information. The table's architecture to store URLs and the relative data type is as shown in table 1. Each URL occupies a row. The table's architecture to store documents and the relative data type is as shown in Table 2. Each document occupies a row.

Field	Data Type
HOST_ID	NUMBER(6)
PROT	VARCHAR2(10)
HOST	VARCHAR2(60)
PATH	VARCHAR2(250)
OLDPATH	VARCHAR2(250)
FLAG	CHAR(1)
PRIORITY	NUMBER(2)
DEPTH	NUMBER(2)
FID	NUMBER(8)

Table 1. The table's architecture to store URLs and relative Data Type

Field	Data Type
FID	NUMBER(8) constraint pk_fid primary key,
STATUS	NUMBER(1),
HOST_ID	NUMBER(6),
PROT	VARCHAR2(10),
HOST1	VARCHAR2(60),
PATH	VARCHAR2(255),
TIME	DATE,
LTIME	DATE,
NO_REFS	NUMBER(5),
NO_HTML	NUMBER(5),
NO_IMGS	NUMBER(5),
S_FORM	NUMBER(1),
FSIZE	NUMBER(8),
TITLE	VARCHAR2(255),
SUBJECT	VARCHAR2(255),
TEXT	LONG

Table 1. The table's architecture to store the documment and relative information

2.2 Index

The index is an optional structure associated with tables and clusters. It provides a faster access path to table data. Properly used, indexes are primary means of reducing disk I/O. The absence or presence of an index dose not require a change in the wording of any SQL statement. An index is

merely a fast path to the data, it affects the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value.
Indexes are logically and physically independent of the data in the associated table. They can be created or dropped anytime without any effect on the base tables or other indexes. If an index is dropped, all applications continue to work. However, access of previously indexed data might be slower.

Once created, an index is automatically maintained and used by Oracle. Changes to data, such as inserting new rows, updating rows, or deleting rows, are automatically reflected in all relevant indexes with no additional action by users. The presence of many indexes on table decreases the performance of updates,deletes and inserts since the indexes associated with the table must also be updated.

In order to increase the performance of update, delete and insert to document table, we adapt batch index mode to the text of document. That is to say, we index the text of documents after loading all documents. For those non_text columns we adapt real-time index, (e.g URL's id and document's id).
We adapt Oracle ConText Option to create index for the text of document. First, we define a policy for the text column of document by calling CTX_DDL.CREATE_POLICY procedure and specify two required parameters: policy name and fully qualified column name for the policy. The example is as shown below:

```
execute ctx_ddl.create_policy('search','search.docs.text')
```

Here, a policy named SEARCH is created for the TEXT column in the DOCS table owned by SEARCH.

After that, we create a text index for the text column by calling the CTX_DDL.create_INDEX procedure. The example is as shown below:

```
execute ctx_ddl.create_index('search')
```

Here, an index is created for all the documents in the column specified in the SEARCH policy.

2.3 Search

After a text index is created for a column, ConText Server with the Query personality can process text queries for the column.

Oracle ConText Option is an option to Oracle. It provides powerful search, retrieval, and viewing capabilities for text stored in an Oracle database. These capabilities are implemented in Oracle using a ConText Server process that complements the Oracle Server processes and a text index, created by ConText Option, fro each database column in which text is stored.

ConText Option also provides in-depth liguistic analysis for English-language text stored in the database, in the form of Linguistic Service. These Linguistic Services produce linguistic output which can be used to create advanced, language-intelligent search and viewing capabilities for your text.

With ConText Option, we can use standard SQL and Oracle's procedure extenstion to SQL, PL/SQL, to incorporate the following text and linguistic functionality into new or existing applications:

a) advanced text searching and retrieval, such as:
-- queries based on content
-- linguistic queries based on main themes

b) advanced text viewing, such as:
-- query term highlighting
-- theme lists
-- text summaries called Gists

We use PL/SQL package to implement the queries to the text of documents by calling CTX_QUERY.CONTAINS procedure. The example is as shown below:

```
ctx_query.contains('SEARCH', 'keyword','QUERY_TEMP',)
```

Here, SEARCH is the name of policy created for the text column.

Keyword is the queried expression., it defines the search criteria specified by the user. A query expression is a combination of query terms (words and phrases) and components, such as operators, that allow users to specify exactly which documents are to be retrieved.

There are several search options available for querying text, including:

a) Exact word or phrase

b) Associations
-- Stem of a word or phrase
-- Fuzzy match of a word or phrase which allows for misspelling
-- Words that sound similiar to each other
-- Words specified as LIKE expressions
The expansion operators is shown in table 3.

Operator	Symbol	Description
STEM	$	Expands the list of words to include all words having the same stem or root word
FUZZY	?	Expands the list of words to include all words with similar spelling

SOUNDEX	!	Expands the list of words to include all words that sound the same

Table 3. The expression operators

c) Proximity

Search for words or phrases that appear close to one another in a document. Documents where the words or phrases are close together will score higher than those where the terms are farther apart.

d) operators

Any of these options can be logically combined. Logical operators combine the term in a query expression. All single words and phrases may be combined with logical operators. When query term are combined, neither the number of spaces around the logical operator, nor the order of the query terms is significant, with exception of the MINUS operator(i.e., ‘ A minus B ‘ is different from ‘ B minus A ‘). All operators is as shown in table 4:

Operator	**Symbol**	**Equivalent**
AND	&	and
OR	|	or
ACCUMULA	,	accum
MINUS	-	minus

table 4. Logical operators and relative meanning

CTX_TEMP is the results table created before the query is performed. The result table is always called hitlist table, it must have the structure illustrated by the table 5.

Field	**Data Type**
TEXTKEY	VARCHAR2(64)
SCORE	NUMBER
CONID	NUMBER

Table 5. The architecture to the query results and relative data type

Our search system displays the document list page by page, including title, size, last update time and subject after handling the search request. If user wants to read the detail of a document, he can click on the document using mouse and the whole document will be displayed with the keywords are highlighted.

We implemented viewing document in Pro*C by calling the procedure CTX_QUERY.HIGH-

LIGHT. This procedure fetches the document, parses the query, identifies the matching terms and insert results into the HIGHLIGHT_TEMP table.
The example is as shown in the table 6.

Field	Data Type
ID	NUMBER
OFFSET	NUMBER
LENGTH	NUMBER
STRENGTH	NUMBER

Table 6. HIGHLIGHT_TEMP table's column and relative data type

According to the offset and length of the keywords in the document, we form the highlight for the keywords in HTML format.

3. Web Interface

We provide user web interface for entering the query string. User can type any keyword expression according to the search example given by the table in the interface. The main Web Interface is as shown in the Figure 2.
After accepting the user's query string, the interface calls PL/SQL package by using WOW gateway, returns the result list. The interface is illustrated as in the Figure 3.
Then user can click on the document to view the document, and it returns the detail of the document with highlighted the keywords. This is illustrated as in the Figure 4.