Temporal Ranking for Fresh Information Retrieval

Nobuyoshi Sato
Dept. of Information and Computer Sciences
Toyo University
Kawagoe, Saitama, Japan
jju@ds.cs.toyo.ac.jp

Minoru Uehara
Dept. of Information and Computer Sciences
Toyo University
Kawagoe Saitama Japan
uehara@toyo.ac.jp

Yoshifumi Sakai
Graduate School of Agricultural Sciences
Tohoku Univeristy
Sendai Japan
sakai@biochem.toyo.ac.jp

Abstract

In business, the retrieval of up-to-date, or fresh, information is very important. It is difficult for conventional search engines based on a centralized architecture to retrieve fresh information, because they take a long time to collect documents via Web robots. In contrast to a centralized architecture, a search engine based on a distributed architecture does not need to collect documents, because each site makes an index independently. As a result, distributed search engines can be used to retrieve fresh information. However, fast indexing alone is not enough to retrieve fresh information, as support for temporal information based retrieval is also required. In this paper, we describe temporal information retrieval in distributed search engines. In particular, we propose an implementation of temporal ranking.

1. Introduction

In our information-intensive society, it is important for us to know what information was up-to-date, or fresh, at a certain point in time. However, since search engines having a centralized architecture, such as Google, require an enormous amount of time to collect all documents in a network, it is difficult to retrieve fresh information using them even in the present. In order to realize fresh information retrieval, we have developed Cooperative Search Engine (CSE)[2]. CSE has a distributed architecture, and hence does not have to collect all the documents in the network. Each local site acts as a local search engine for documents in the site, and each local index for the documents is updated every few minutes. For this reason we can retrieve fresh information via CSE.

It is a notable characteristic of CSE that retrieval results can immediately reflect when the appearance of a new document or editing of an existing document occurs. However, since the retrieval results contain not only fresh documents but also stale documents, it is not easy to determine which documents include fresh information. In order to solve this problem, we try to implement a function in CSE for selecting documents that were fresh at a point in time arbitrarily specified by user.

This paper is organized as follows. In section 2, we survey temporal databases and temporal information retrieval. In section 3 we describe CSE and in section 4 we define temporal information in CSE. We describe the implementation of temporal information retrieval in CSE in section 5 and evaluate it in section 6. Finally, we end the paper with some conclusions.

2. Temporal Information Retrieval

The value of information is determined by the ratio of the number of information consumers who want the information to the number of information providers who have the information. If the number of information providers increases then the information value decreases. Information that is known to everyone is called common knowledge. According to Shannon's information theory, information is entropy. In other words, information creates a system from chaos, although the system is temporary and will soon diffuse. Information value is at its highest when the system is first created. Therefore, the freshest information is the most valuable. Information retrieval is the process of finding valuable information, and in this sense, fresh information retrieval is extremely important.

It is clear that fresh information retrieval is a special type of temporal information retrieval. Temporal information retrieval is the process of extracting time-varying information. A document may be modified any time after it is created, and hence a document consists of time-varying information. For example, a word which was included in a document before modifying often is not included in a document after modifying. Therefore, time-varying information must be retrieved with the time specified. This is quite natural and such temporal information retrieval is available for digital libraries.

2.1 Temporal Database

Although information retrieval is not data retrieval, the theoretical background of temporal information retrieval is in temporal databases. A temporal database is a database to which a time interval can be specified as a query. The time interval is based on temporal interval logic proposed by J. F. Allen[14]. Therefore, temporal information retrieval must support time intervals as part of a query. In a temporal

database, the unit of time is the chronon.

The granularity of a chronon is selected from year, month, day, hour, minute, and second.

Assume that there are time points t_1, t_2, t_2', t_3, t_3', t_4 (t_i<t_{i+1}, t_i=t_i'). Also assume that $[t_i,t_j]$(i<j) is a time interval, where start($[t_i,t_j]$)=t_i, and end($[t_i,t_j]$)=t_j. The following relations exist among time points X and Y, and time intervals A and B.

X* before *Y: X<Y, e.g. t_1 before t_2

X* after *Y: X>Y, e.g. t_2 after t_1

X* simultaneous-with *Y: X = Y, e.g. t_2 simultaneous-with t_2'

X* in *A: start(A)≤X≤end(Y), e.g. t_2 in $[t_1,t_3]$

A* before *B: end(A)<start(B), e.g. $[t_1,t_2]$ before $[t_3,t_4]$

A* meets *B: end(A)=start(B), e.g. $[t_1,t_2]$ meets $[t_2',t_3]$

A* overlaps *B: start(B)<end(A)<end(B) ∩ start(A)<start(B), e.g. $[t_1,t_3]$ overlaps $[t_2,t_4]$

A* starts *B: start(A)=start(B), e.g. $[t_2,t_3]$ starts $[t_2',t_4]$

A* during *B: start(A)>start(B) ∩ end(A)<end(B), e.g. $[t_2,t_3]$ during $[t_1,t_4]$

A* finished *B: end(A)=end(B) ∩ start(A)>start(B), e.g. $[t_2,t_3]$ finishes $[t_1,t_3']$

A* after *B: start(A) > end(B), e.g. $[t_3,t_4]$ after $[t_1,t_2]$

A* met-by *B: start(A)=end(B), e.g. $[t_2,t_3]$ met-by $[t_1,t_2']$

A* overlapped-by *B: start(B)<start(A)<end(B) ∩ end(B)<end(A), e.g. $[t_2,t_4]$ overlapped-by $[t_1,t_3]$

A* started-by *B: start(A)=start(B) ∩ end(A)>end(B), e.g. $[t_2,t_4]$ started-by $[t_2',t_3]$

A* contains *B: start(A)<start(B) ∩ end(A)>end(B), e.g. $[t_1,t_4]$ contains $[t_2,t_3]$

A* finished-by *B: end(A)=end(B) ∩ start(A)<start(B), e.g. $[t_1,t_3]$ finished-by $[t_2,t_3']$

A* cotemporal *B: start(A)=start(B) ∩ end(A)=end(B), e.g. $[t_2,t_3]$ cotemporal $[t_2',t_3']$

In a temporal database, there are 2 kinds of times: valid times and transaction times. Valid times concern facts that are true in modeled reality. Transaction times concern facts that are current in the database.

In general, a valid time DB stores only fresh data, whereas a transaction time DB stores the complete history of the data. A bitemporal DB supports both kinds of data.

2.2 The Concept of Temporal Information Retrieval

In this paper, temporal information retrieval is defined as determining whether or not a document exists at a time point or in a time interval. This is in contrast to whether or not the content of a document includes the specified time. For example, assume that a document containing the text "In 2002, the FIFA World Cup will be held in Korea and Japan" was written in 1998. In the former case, this document would be retrieved with the query, 1998 and (Korea or Japan). In the latter case, this document would be retrieved with the query, 2002 and (Korea or Japan). The number 1998 in the former case is the modified time of the document. The number 2002 in the latter case is a keyword in the text of the document. This latter type of retrieval is classified as a query expansion or a numerical query. We discuss temporal information retrieval in the former sense.

Assume that a document always contains facts. In this case, a fact in temporal information retrieval means the existence of the document. Valid time is the time when the document exists in the real world, and transaction time denotes the time when the document is indexed.

The lifetime of a document depends on the document model, and there are two kinds of models. The first is the immutable model, in which the lifetime of a document is equivalent to the lifetime of the information. The information is the content of the document, and when a document is modified, the information is also changed. Therefore, an old document is deleted and a new document is created at every modification time. The second type of model is the mutable model, in which the modification of a document is allowed. In this model, when a document is modified, the content of the document is changed but the document itself is not changed. So, in the mutable model, a document exists from the time it is created to the time it is deleted, although its content may change multiple times. In the immutable model, a document exists only from one modification time to another. From the viewpoint of the users the retrieval result, with the exception of time, is not dependent on the document model. However, in the immutable model, the retrieval result is based on the modification time, whereas in the mutable model, it is based on the creation time.

There are several possible interpretations of created time, modified time and deleted time. Assume that someone had information at time t_1, he wrote it into a document at t_2, he published the document at t_3, and the document was indexed by a search engine at t_4. It is important to determine what time corresponds to the origin of the information. In principle, the information is created at t_1. However, it is hard to prove this fact and it is impossible to retrieve it. The time t_2 is determined by outside factors. In addition, it may not be possible for everyone to publish a web document without changing the timestamp, so, t_2 is not a good measure. The time t_3 is the published time when the document is available on the web. However, it is difficult to retrieve the document at precisely t_3. In fact, we can retrieve the document after t_4. Ideally, t_4 should be nearly equal to t_3. In centralized search engines, because $t_4 - t_3$ is greater than $t_3 - t_2$, t_2 is used instead of t_4. However, in distributed search engines, because $t_4 - t_3$ is very small, t_4 is used for the purpose of temporal information retrieval. In such a case, the valid time is equivalent to the transaction time.

There are two kinds of temporal queries in temporal information retrieval. One is an interval query which retrieves documents existing in an interval of time. The other is a point query which retrieves documents existing at a certain time point. An interval query is also called a time slice query. A temporal query is used in conjunction with a keyword query. The retrieval results include not only the content of the documents, but also the created time and the

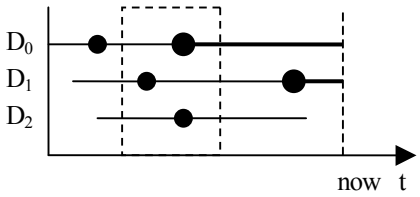

Figure 1. Temporal Information Retrieval

modified time.

The targets of a temporal query are the lifetime interval and the modified time point of the document. In a temporal query, temporal relations mentioned in section 2.1 may be specified.

2.3 Fresh Information Retrieval

In order to realize fully temporal information retrieval, it is necessary to store the complete history of every document's modification, however this has huge storage requirements. So instead, we introduce fresh information retrieval as a practical substitute, which retrieves the last modified versions of current documents.

Temporal information retrieval is the retrieval of documents that exist during a time interval. Fresh information retrieval is not the retrieval of documents that have current content, but to retrieve current documents which exist with content during a time interval. With fresh information retrieval, huge storage is unnecessary because only the last modified version of a document is stored. Also, fresh information retrieval supports all the functions of temporal information retrieval except that the retrieved document is the current version. In section 2.1, we described that a valid time DB stores only current versions of documents. In this sense, fresh information retrieval is valid time information retrieval.

We illustrate 3 kinds of information retrieval in Fig. 1. In this figure, there are 3 documents D_0, D_1 and D_2, and the black dots represent modification events. In non-temporal information retrieval, documents which exist at the current point in time are retrieved. In Fig. 1, D_0 and D_1 are retrieved by non-temporal information retrieval. D_2 is not retrieved because it is deleted. In fresh information retrieval, D_0 and D_1 are retrieved in the same way as in non-temporal information retrieval. However, D_0 is retrieved with the temporal query shown as the dashed rectangle in Fig. 1. Non-temporal information retrieval does not support such a query. Finally, in fully temporal information retrieval, all documents D_0, D_1, and D_2 may be retrieved with any temporal query. For example, D_0 exists as 3 versions separated by two modifications.

3. Cooperative Search Engine

First, we explain a basic idea of CSE. In order to minimize the update interval, every web site basically makes indices via a local indexer. However, these sites are not cooperative yet. Each site sends the information about what

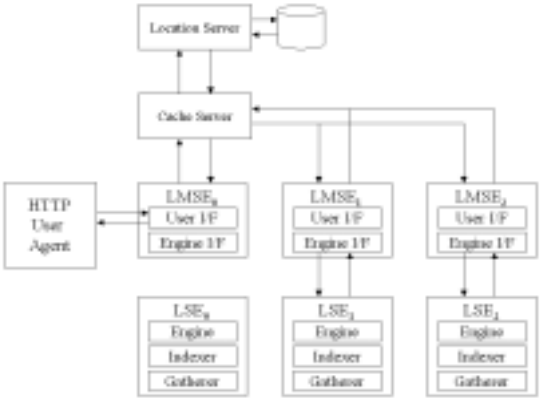

Figure 2. The overview of CSE

(i.e. which words) it knows to the manager. This information is called Forward Knowledge (FK), and is Meta knowledge indicating what each site knows. FK is the same as FI of Ingrid. When searching, the manager tells which site has documents including any word in the query to the client, and then the client sends the query to all of those sites. In this way, since CSE needs two-pass communication at searching, the retrieval time of CSE becomes longer than that of a centralized search engine.

CSE consists of the following components (see Figure 1).

- Location Server (LS): It manages FK exclusively. Using FK, LS performs Query based Site Selection described later. LS also has Site selection Cache (SC) which caches results of site selection.
- Cache Server (CS): It caches FK and retrieval results. LS can be thought of as the top-level CS. It realizes "Next 10" searches by caching retrieval results. Furthermore, it realizes a parallel search by calling LMSE mentioned later in parallel.
- Local Meta Search Engine (LMSE): It receives queries from a user, sends it to CS (User I/F in Figure 2), and does local search process by calling LSE mentioned later (Engine I/F in Figure 2). It works as the Meta search engine that abstracts the difference between LSEs.
- Local Search Engine (LSE): It gathers documents locally (Gatherer in Figure 2), makes a local index (Indexer in Fig. 2), and retrieves documents by using the index (Engine in Figure 2). In CSE, Namazu[1] can be used as a LSE. Furthermore we are developing an original indexer designed to realize high-level search functions such as parallel search and phrase search.

Namazu has widely used as the search services on various Japanese sites.

Next, we explain how the update process is done. In CSE, Update I/F of LSE carries out the update process periodically. The algorithm for the update process in CSE is as follows.

1. Gatherer of LSE gathers all the documents (Web

pages) in the target Web sites using direct access(i.e. via NFS) if available, using archived access(i.e. via CGI) if it is available but direct access is not available, and using HTTP access otherwise.
Here, we explain archived access in detail. In archived access, a special CGI that provides mobile agent place functions is used. A mobile agent is sent to that place. The agent archives local files, compresses them and sends back to the gatherer.

2. Indexer of LSE makes an index for gathered documents by parallel processing based on Boss-Worker model.
3. Update phase 1: Each $LMSE_i$ updates as follows.

3.1. Engine I/F of $LMSE_i$ obtains from the corresponding LSE the total number N_i of all the documents, the set K_i of all the words appearing in some documents, and the number $n_{k,i}$ of all the documents including word k, and sends to CS all of them together with its own URL.

3.2. CS sends all the contents received from each $LMSE_i$ to the upper-level CS. The transmission of the contents is terminated when they reach the top-level CS (namely, LS).

3.3. LS calculates the value of $idf(k) = \log(\sum N_i / \sum n_{k,i})$ from $N_{k,i}$ and N_i for each word k.

4. Update phase 2: Each $LMSE_i$ updates as follows

4.1. LMSEi receives the set of Boolean queries Q which has been searched and the set of *idf* values from LS.

4.2. Engine I/F of $LMSE_i$ obtains from the corresponding LSE the highest score $\max_{d \in D} S_i(d,q)$ for each $q \in \{Q, K_i\}$, $S_i(d,k)$ is a score of document d containing k, D is the set of all the documents in the site, and sends to CS all of them together with its own URL.

4.3. CS sends all the contents received from each $LMSE_i$ to the upper-level CS. The transmission of the contents is terminated when they reach the top-level CS (namely, LS).

Note that the data transferred between each module are mainly used for distributed calculation to obtain the score based on the *tf*idf* method. We call this method the distributed *tf*idf* method. The score based on the distributed *tf*idf* method is calculated at the search process. So we will give the detail about the score when we explain the search process in CSE.

For the good performance of the update process, the performance of the search process is sacrificed in CSE. Here we explain how the search process in CSE is done.

1. When $LMSE_0$ receives a query from a user, it sends the query to CS.
2. CS obtains from LS all the LMSEs expected to have documents satisfying the query.
3. CS sends the query to each of all LMSEs obtained.
4. Each LMSE searches documents satisfying the query by using LSE, and returns the result to CS.
5. CS combines with all the results received from LMSEs, and returns it to $LMSE_0$.
6. $LMSE_0$ displays the search result to the user.

.Here, we describe the design of scalable architecture for the distributed search engine, CSE.

In CSE, at searching time, there is the problem that communication delay occurs. Such a problem is solved by using following techniques.

- Look Ahead Cache in "Next 10" Search[3]
  To shorten the delay on search process, CS prepares the next result for the "Next 10" search. That is, the search result is divided into page units, and each page unit is cached in advance by background process without increasing the response time.
- Score based Site Selection (SbSS)[4]
  In the "Next 10" search, the score of the next ranked document in each site is gathered in advance, and the requests to the sites with low-ranked documents are suppressed. By this suppression, the network traffic does not increase unnecessarily. For example, there are more than 100,000 domain sites in Japan. However, by using this technique, about ten sites are sufficient to requests on each continuous search.
- Global Shared Cache (GSC)[5]
  A LMSE sends a query to the nearest CS. Many CS may send same requests to LMSEs. So, in order to globally share cached retrieval results among CSs, we proposed Global Shared Cache (GSC). In this method, LS memories the authority CS_a of each query and tells CSs CS_a instead of LMSEs. CS caches the cached contents of CS_a.
- Persistent Cache(PC)[6]
  There is at least one CS in CSE in order to improve the response time of retrieval. However, the cache becomes invalid soon because the update interval is very short in CSE. Valuable first page is also lost. Therefore, we need persistent cache, which holds valid cache data before and after updating. In this method, there are two update phases. At first update phase, each LMSE sends the number of documents including each word to LS, and LS detects idf of each word. At second update phase, preliminary search is performed using new idfs in order to update caches.
- Query based Site Selection(QbSS)[7][8]
  CSE supports Boolean search based on Boolean formula. In Boolean search of CSE, the operations "and", "or", and "and-not" are available. Let S_A and S_B be the set of target sites for search queries A and B, respectively. Then, the set of target sites for queries "A and B", "A or B", and "A and-not B" are $S_A \cap S_B$, $S_A \cup S_B$, and S_A, respectively. By this selection of the target sites, the number of messages in search process is saved.

These techniques are used as follows:

if *the previous page of "Next 10" search has been already searched*

LAC
else if *query does not contain "and" or "and-not"*
SbSS
else if *it has been searched since index was updated*
GSC
else if *it has been searched once*
PC
else *// query is new*
QbSS
fi

4. Temporal Information Retrieval in CSE

4.1 Temporal Query

Here, we describe the temporal queries used to support the retrieval of temporal information. CSE currently supports Boolean queries for keywords, and temporal queries in addition to keyword queries. Temporal queries are used to select documents existing at certain times or within certain time intervals.

A temporal query is an expression of a time point or a time interval. First, we define a time point expression. Several conventional search engines can retrieve documents modified in some days or some months. However, this level of granularity is not sufficient for retrieving fresh information. A fresh information retrieval system has to retrieve documents modified within a matter of minutes at least. CSE updates the index within a few minutes independent of the scale of the system. In the near future, we expect to allow retrieval in real time, which is ideal for the purpose of fresh information retrieval. Therefore, we employ the second as the granularity of a chronon.

A computer stores time as an integer which is represented as the number of seconds after 1970-01-01 00:00:00 GMT. However, it is not natural for a human to count time using only seconds, so in this paper we represent time as the following expression.

Y/*M*/*D*/*h*/*m*/*s*

Here, *Y* is the year in A.D., *M* is the numerical month (1-12), *D* is the day in a month (1-31), *h* is the hour (0-23), *m* is the minute(0-59), *s* is the second(0-59). If each granularity is omitted, it denotes an initial value. For an example, *Y* is *Y*/1/1/0/0/0.

Furthermore, a time which is prefixed with a minus sign denotes the difference from the current time.

-*Y*/*M*/*D*/*h*/*m*/*s*

For example, -1/6 is a year and 6 months ago. If the accepted temporal query is negative, it is added to the current time. A negative temporal query is provided for the user's convenience.

Next, we define the attributes of a document and their symbols as time point variables.

/c the created time of the document
/e the effective modified time of the document
/m the last modified time of the document
/now the current time

Here, the effective modified time of the document denotes the last modified time where the content of the version is nearly equal to that of the current version. We will describe how to calculate /e in section 4.2. In the immutable document model, /m is used, and in the mutable document model, /c is used. The relationship of /c≤/e≤/m≤/now is always true.

The following queries exist concerning time points t_1 and t_2.

$t_1 < t_2$: t_1 before t_2
$t_1 > t_2$: t_1 after t_2
$t_1 = t_2$: t_1 simultaneous-with t_2

Here, time point queries are compared with each other in the smallest granularity even if they form an elliptical representation.

A time interval is represented as $[t_1,t_2]$ using two time points t_1 and t_2. If a time point T is included in $[t_1,t_2]$ ($T \in [t_1,t_2]$), $t_1 \leq T \cap T < t_2$. Although $[t_1,t_2)$ is mathematically more accurate compared with $[t_1,t_2]$, $[t_1,t_2]$ is easy for us to understand. In Allen's temporal interval logic, which lacks the concept of a time point, it is not clear whether both edges of the time interval are included in the range of the time interval or not. In our system, we allow an elliptical representation of a time interval such as $[T] = [T,T+1]$, where $T+1$ denotes the increment of the smallest explicit granularity, e.g. [2000]=[2000,2001], [2002/1/31]=[2002/1/31,2002/2/1]. The lifetime of the document is represented as [/c,/now].

As mentioned in section 2.1, there are a large number of relationships between Allen's time intervals. However, they can all be reduced to relationships between time points and the functions giving the start point and the end point of the time interval. For this reason, CSE does not support interval queries but only point queries.

Next, we discuss whether a temporal query is mixed with a keyword query or not. In the case of mixing, the semantics of a query is simple but its implementation is complex. Conversely, without mixing, the semantics of a query is complex but it can be implemented easily. For example, we can use the following query if mixing is allowed.

"FIFA World Cup" and (("Korea" or "Japan") and (/c in [2002])) or ("France" and (/c in [1998]))

This query searches for both documents that describe the World Cup held in Korea and Japan in 2002 and documents that describe the World Cup held in France in 1998.

On the other hand, if mixing is not allowed, the following query could be used.

"FIFA World Cup" and ("Korea" or "Japan" or "France")
/c in [2002] or /c in [1998]

Here, the relationship between keyword query and temporal query is conjunctive. This query searches for documents that describe both the World Cup of France and the World Cup of Korea and Japan in 1998 or 2002. In the latter method, a document describing Korea and Japan in 1998 and another document describing France in 2002 may both be retrieved. Therefore, we employ the former method.

Temporal query *TQ* is represented with BNF as follows:

TQ : *Q* | *TQ* or *TQ* |

TQ and *TQ* | *TQ* and *TC* | *TC* and *TQ* |
TQ not *TQ* | *TQ* not *TC*

Q : K | Q and Q | Q or Q | Q not Q

TC : $T_v > T_c$ | $T_v < T_c$ | $T_v = T_c$ | $T_v \leq T_c$ | $T_v \geq T_c$ |
T_v in $[T_c]$ | T_v in $[T_c, T_c]$ |
TC or *TC* | *TC* and *TC*

Here, K is a keyword, Q is a Boolean expression of keywords, T_v is a time point variable, T_c is a time point constant, and TC is a temporal query. Note that TC alone cannot be the temporal query TQ. This is because all documents may be selected if only TC is the query, and such retrieval is not useful. Especially in distributed search engines, a traffic overload may occur because sites are not selected. TC is used to select from the result of Q using a temporal condition.

The time in a temporal query is not the time interval where information is current but the time point of the origin of information. Therefore, the query =/now cannot match any document. The query </now can match the same documents as a non-temporal information retrieval.

4.2 Content based Freshness

For a user who wants to know what was fresh at a certain point in time, it is useful to display a list of documents that were fresh at that time. However, selecting documents according to the last modification time recorded by the file system is not appropriate because even if the last change to a document was only the correction of a slight typographical error, the document is regarded as having new content at that modification time. On the other hand, adopting the time when each document was published on the network is also undesirable because we cannot recognize that a document was fresh at the point in time when the content of the document was completely changed.

These shortcomings arise from the policy of treating the freshness of a document without taking into account the change of the meaning of the content. Unfortunately it is difficult to determine whether the content of a document has largely changed or not. In this paper, we propose an alternative method of determining the change in content of a document, by using the change in *TF*IDF*s for keywords appearing in it. In CSE, a retrieval result is displayed to the user as a list of documents ranked according to *TF*IDF* for the retrieval query. In the same way as other search engines adopting *TF*IDF* ranking, if an OR search for all keywords is requested to CSE, all documents are ranked according to the largest *TF*IDF* for a keyword appearing in each document, which implies that we can think of a document as containing information regarding the keyword for which *TF*IDF* is the largest. Therefore, when the keyword having the largest *TF*IDF* is changed by editing a document, the content of the document is thought of as having changed, and the document is then ranked according to the keyword that has the largest *TF*IDF* after the change. The proposed method for determining whether or not the content of a document has changed obeys this policy of *TF*IDF* ranking.

The concrete algorithm for the method is as follows:

For any time,
  if the keyword that has the largest *TF*IDF* in the document has changed, then
    update the time stamp of the document being fresh to be the current time.

4.3 Temporal Ranking

Ranking means sorting retrieved results. Conventional search engines sort retrieved results in the descending order of document scores. However, in temporal information retrieval, temporal ranking is required. In temporal ranking, a temporal search engine sorts retrieved results in order of document time. Here, assume that ranking method is independent on Boolean formula of keywords in a query.

In temporal ranking, QbSS and SbSS work well as same as they work well in score based ranking. These effects are summarized as table 2. In first column of table 2, there are two kinds of ranking order: "newer" and "older". Here, top item is the newest one in newer order, and it is the oldest one in older order. In second column, there are two kind of basic time point queries: $T_v < T_c$, and $Tv > T_c$. The third column, "Case" shows the relation of Tc in a query to total time interval [min, max] of a server. Total time interval includes last modified times of all documents in a server. Finally, in fourth column "effect," several site selection techniques which work well are listed. When QbSS works well, the site is ignored by QbSS. SbSS means that SbSS works well. PC(Persistent Cache) means that SbSS does not work well but PC may work. SbSS works well if max is the time of top item in the newer order or if min is the time of top item in the older order. A query is sent to the server iff either SbSS or PC.

SbSS is a key technique for scalability. SbSS does not work well if non-temporal query includes either AND or AND-NOT. However, in temporal query, SbSS may work well even if a temporal query includes AND and AND-NOT. This is because complex time interval query can be reduced to the range of one dimension of time. For an example, ORed time interval query $\cup_{i=1..n}[s_i,e_i]$ is reduced to [min s_i,

Table 2. The Effect of Site Selection

Order	Query	Case	Effect
Newer	Tv < Tc	max < Tc	SbSS
		min < Tc < max	PC
		Tc < min	QbSS
	Tv > Tc	max < Tc	QbSS
		min < Tc < max	SbSS
		Tc < min	SbSS
Older	Tv < Tc	max < Tc	SbSS
		min < Tc < max	SbSS
		Tc < min	QbSS
	Tv > Tc	max < Tc	QbSS
		min < Tc < max	PC
		Tc < min	SbSS

max e_i], and ANDed time interval query $\cap_{i=1..n}[s_i,e_i]$ is reduced to[max s_i, min e_i]. In this way, all time interval query can be reduced to simple time point query in table 2. Therefore, SbSS is efficient in temporal ranking. However, SbSS does not work well if both temporal queries and non-temporal queries are combined. From such a point of view, temporal query should not be used with non-temporal query. Although SbSS is not effective, PC may work well. This is because PC works well if the query has already been retrieved once.

5. Implementation

In this section, we describe the implementation of fresh information retrieval.

In CSE, LMSE searches for documents by calling LSE. LSE must support *TF* based scoring (not *TF***IDF*). Namazu, one of the most popular small search engines in Japan supports *TF* scoring. We assumed Namazu is used as the implementation of LSE in our system.

LSE constructs an index when updating occurs. Here, LSE changes *TF* of an index even if documents are slightly modified. This is the original behavior of LSE.

LMSE has yet another index. After LSE has finished updating LSE's index, LMSE extracts *TF* values from each document in LSE's index, and compares each *TF* value from LMSE's index and LSE's index. If they are different, LMSE copies the *TF* value of the document from LSE to LMSE's index, and changes the publish timestamp of a document to be the time LSE began the updating. Finally, LMSE extracts the highest scores of each word and range of timestamps (oldest and latest) of each document, and sends them to LS. Since LSE is used to search, slight changes to documents are reflected in their scores. However, the timestamp is replaced by the time recorded by LMSE.

If a query includes a temporal expression, Query based Site Selection (QbSS)[7][8] is also used to select search target sites. Since LS has only the latest timestamps, LS cannot select sites. However, it is effective for fresh information retrieval, which is the main purpose of CSE.

LMSE descends a query recursively, and requests a single keyword expression from LSE. LSE returns a result which is sorted in *TF* order. LMSE multiplies *IDF*, and carries out a set operation, selecting by temporal condition. The search results are sorted in order of scores by a specified ranking method. CS does not share the cache queues for different ranking methods.

6. Evaluations

At first, we will show that the distributed search engine can retrieve fresh information. In paper[2], we compared update intervals in the same document set between CSE and a centralized search engine which used Namazu and wget. A centralized search engine spent 2 hours and 20 minutes, whereas CSE finished in a few minutes. CSE did not fail to search for fresh information within the bounds of these few minutes.

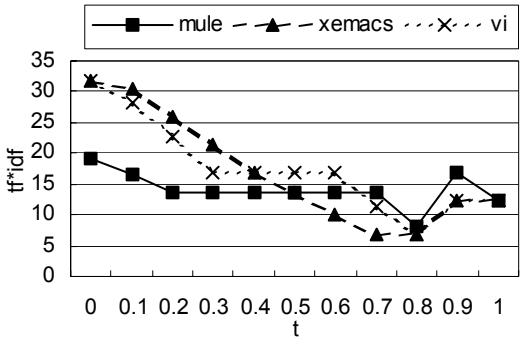

Figure 3. *TF*IDF* max scores

Assume that there are three documents, *A*, *A*' and *A*", which have similar subjects, and a fourth document, *B*, on a different subject. Let the documents which are mixed be *A* and *A*', *A*", *B*, in the ratio of *t*:1−*t* as $tA+(1-t)A'$, $tA+(1-t)A''$, $tA+(1-t)B$. Fig. 3 shows the relationship between *t* and the maximum values of *TF***IDF*. Here, the subjects of *A*, *A*', *A*" and *B* are emacs, mule, xemacs and vi respectively. The order of closeness to the subject of emacs is mule < xemacs < vi. Words which have the maximum *TF***IDF* value in each document are changed at t=2 in mule, which has a similar subject to emacs. In vi, which has quite a different subject, the maximum *TF***IDF* word changed at t=3. Therefore, it will be judged that the content was changed if 20 to 30% of documents were changed, when the variation of the content is detected by the maximum value of *TF***IDF*.

7. Related Works

There are two types of temporal information retrieval: retrieving documents by time and displaying documents in the order of time. Namazu[1], Goo, Infoseek, NAVER[11], Google and so on can be used to search documents by time. Namazu searches HTML documents with HTTP headers and e-mail like documents by using a regular expression involving time. Since these documents have a date: field in their header, they can easily be searched by time. However, normal HTML documents without headers have no date: fields. In HTML documents with a header, the date: field often denotes the time that they were downloaded. For this reason, Namazu can not search web documents by time.

In Goo, a user can select before/after a particular date. Goo searches for the newest information since Goo does not distinguish between different versions of a document. However, searching documents by date is not efficient for fresh information retrieval. Searching by second, or at the most by minute, is required.

In Infoseek, a user can also select before/after a particular date, and Infoseek supports searching by a range of dates.

NAVER supports specifying a range of months in document search mode which searches for non-HTML documents such as MS Word, Excel files, PDF and so on. However, specifying a range of months is completely unsuitable. Furthermore, NAVER does not support

specifying a particular date or month.

In Google, a user can select "past 3, 6, 12 months" in Advanced Search mode. However, this is not as efficient as NAVER.

Among those mentioned above, Infoseek is most similar to fresh information retrieval, however the freshness is insufficient because Infoseek only supports specifying documents by date.

Namazu, FreshEye and NAVER display search results in order of time. They can also display results in increasing or decreasing order. Other search engines such as Yahoo, AltaVista, Excite and Lycos do not support searching by time.

In the field of databases, there is much work regarding temporal database management[12]. The Valid Web[13] realizes temporal retrieval by specifying the valid time of web documents using XML. However, no HTML documents are able to specify a valid time.

Although search engines are a kind of database, few experiments have been conducted on retrieving temporal information. One of the reasons is the search engine architecture. The search engines mentioned above all have a centralized architecture. Centralized search engines spend a lot of time gathering documents. Therefore, it is difficult for these search engines to collect temporal information. However, with distributed search engines, almost real-time retrieval is practical since they do not need to gather documents over the network.

A number of distributed search engines exist, such as Whois++[9], Harvest[10], GlOSS and so on. Whois++ and Harvest use forward knowledge. Forward knowledge is also used in CSE, however, these systems have no limitation on retrieval response time. CSE realizes regular response time regardless of its scale. In addition, these search engines do not support temporal information retrieval.

7. Conclusions

In this paper, we introduced the concept of temporal information retrieval, and clarified the difference between fresh information retrieval, which is a subset of temporal information retrieval and existing information retrieval. We discussed the necessary conditions for fresh information retrieval, and described an implementation of it in CSE. Also, we proposed an implementation of temporal ranking in CSE.

The following is a list of our future work: verifying the effectiveness of search engines for fresh information retrieval by long–term experiments, and developing a search engine which realizes complete temporal information retrieval.

Acknowledgement

This research was cooperatively performed as a part of "Scalable Distributed Search Engine for Fresh Information Retrieval (14780242)" in Grant-in-Aid for Scientific Research promoted by Japan Society for the Promotion of Science (JSPS).

References

[1] The Namazu Project, "Namazu", http://www.namazu.org/

[2] Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori, "Fresh Information Retrieval using Cooperative Meta Search Engines," In Proceedings of the 16th International Conference on Information Networking (ICOIN-16), Vol.2, 7A-2, pp.1-7, (2002.1.31)

[3] Nobuyoshi Sato, Takashi Yamamoto, Yoshihiro Nishida, Minoru Uehara, Hideki Mori, "Look Ahead Cache for Next 10 in Cooperative Search Engine", in proc. of DPSWS 2000, IPSJ Symposium Series, Vol.2000, No.15, pp.205-210 (2000.12) (in Japanese)

[4] Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori, "Score Based Site Selection in Cooperative Search Engine", in proc. of DICOMO'2001 IPSJ Symposium Series, Vol.2001, No.7, pp.465-470, (2001.6) (in Japanese)

[5] Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori, "Global Shared Cache in Cooperative Search Engine", in proc. of DPSWS 2001, IPSJ Symposium Series, Vol.2001, No.13, pp.219-224, (2001.10) (in Japanese)

[6] Nobuyoshi Sato, Minoru Uehara, Yoshifumi Sakai, Hideki Mori "Persistent Cache in Cooperative Search Engine," MNSA'02

[7] Yoshifumi Sakai, Nobuyoshi Sato, Minoru Uehara, Hideki Mori, "The Optimal Monotonization for Search Queries in Cooperative Search Engine", in proc. of DICOMO2001, IPSJ Symposium Series, Vol.2001, No.7, pp.453-458 (2001.6) (in Japanese)

[8] Nobuyoshi Sato, Minoru Udagawa, Minoru Uehara, Yoshifumi Sakai, Hideki Mori, "Query based Site Selection for Distributed Search Engines", MNSA'03

[9] C. Weider, J. Fullton, S. Spero: "Architecture of the Whois++ Index Service", RFC1913

[10] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwartz: "The Harvest Information Discovery and Access System", 2nd WWW Conference, http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/schwartz.harvest/schwartz.harvest.html

[11] NAVER, http://www.naver.com/

[12] Christian S. Jensen: "Temporal Database Management", Thesis, http://www.cs.auc.dk/~csj/Thesis/

[13] Fabio Grandi, Federica Mandreoli, "The Valid Web: An XML/XSL Infrastructure for Temporal Management of Web Documents," ADVIS2000, pp. 294-303, 2000

[14] J. F. Allen, "Towards a general theory of action and time," Artificial Intelligence, vol. 23, pp.123-154, 1984