2006

Agent-based Similarity-aware Web Document Pre-fetching

Jitian Xiao
Edith Cowan University

10.1109/CIMCA.2005.1631587

Agent-based Similarity-aware Web Document Pre-fetching

Jitian Xiao
School of Computer and Information Science, Edith Cowan University,
2 Bradford Street, Mount Lawley, WA 6050, Australia
E-mail: j.xiao@ecu.edu.au

Abstract

This paper presents an agent-based similarity-aware web document pre-fetching scheme that is built on the similarity-aware web caching architecture. A set of agents are employed to carry out certain duties such as document similarity detection, identification of relevant access patterns, document prediction and network traffic monitoring for document pre-fetching. Preliminary simulations have been conducted to evaluate the proposed scheme, and the results have shown that the new pre-fetching scheme outperforms existing web-document pre-fetching algorithms.

1. Introduction

Web caching is intended to reduce network traffic, server load and user-perceived retrieval latency. Web pre-fetching, which can be considered as "active" caching, builds on regular web caching, minimizing further a web user's access delay. Pre-fetching is a technique that attempts to guess those documents that are likely to be requested when a page leading to them is accessed – success of this technique is measured as a "hit-ratio". However, in such guessing, there is a need for an effective balance to be achieved between user comfort and computational overheads – the extremes are: too little effort applied, resulting in too many on-demand-fetches, while too much effort results in too many pre-fetches. The consequence of either is that of slower response to a user.

Previous work by Xiao [1] in developing pre-fetching predictions between caching proxies and browsing clients was based on measures of similarity between web users established that pre-fetching is capable of increasing the hit-ratio. Xiao's work further established that organization of the cache affects opportunities for successful pre-fetching. In this paper we describe a means of similarity based content management and propose a similarity-aware pre-fetching technique to improve the relative performance of pre-fetching techniques based upon document similarity detection.

Pre-fetch caching in the context of this study will be based upon similarity detection and involve several phases. Similarities will be sought from previously cached documents employing several concurrently applied, but differing, algorithms to detect equivalences of, e.g. broad-content or keywords, images and picture-titles and links contained within pages under scrutiny. Similarities between web documents, having been detected, will then be ranked for candidature to be fetched in anticipation of a user's intentions, and pre-fetching may then proceed.

The rest of the paper is organized as follows. Section 2 describes the similarity measures. Section 3 outlines a similarity-based web cache architecture upon which the pre-fetching is developed. Section 4 proposes an agent-based the similarity-aware web document pre-fetching scheme. Section 5 presents the simulation results, and Section 6 concludes the paper.

2. Similarity measurement and detection

The exercise of measuring similarities among documents follows two main streams: one uses a single relationship between documents[1] or data objects while the other uses multiple relationships. Early research used a single relationship to measure the similarity of data objects. In the original *vector space model* (VSM) [2], "terms" (e.g. key words or stems) were used to characterize queries and documents, yielding a document-term relationship matrix to compute similarities among terms and documents. Such relationships were used to measure the similarity of documents for retrieval and clustering purposes [3]. In the Latent Semantic Index [4], a *singular vector decomposition* (SVD) method is applied to map the

[1] In this paper, a *document* refers to a text document or a web page that may contain text, images and/or pictures.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

document-term matrix into some lower dimensional matrix where each dimension associates with a hidden "concept", where any similarity of text objects is measured by relationships to those "concepts" rather than the keywords they contained.

With the advent of Word Wide Web, relationships with document objects were used to derive similarity [5]. Pitknow [6] applied co-citation to a hyperlink structure to measure any similarity of two web pages. Flesca [7] proposed a method to measure the similarity of two documents that represents the current and the previous version of monitored pages for web change detection.

The approaches introduced above all relied upon a single relationship to measure any similarity of data objects. However, such approaches may run into serious problems when applications require accurate similarity e.g. where multiple types of data objects and relationships must be handled in an integrated manner. Accordingly, in the extended VSM [8], feature vectors of data objects were augmented by adding attributes from objects of other related spaces. Similarity computation is then obtained from calculation on these enhanced feature vectors.

Recently, it has been tried to calculate the similarity of two data objects based upon any similarity of their related data objects [9]. In this paper we define similarity measures of web documents for effective web document caching and pre-fetching. To pre-fetch documents that are of similar topic to the document a user is currently viewing, we derive the content similarity of web documents, ignoring any structural elements, e.g. HTML formatting. For efficacy of on-line pre-fetching, we propose different levels of similarity measures to capture levels of similarity between web documents. In this study, similarities between text documents are measured based on topics, page titles, keywords or page contents or combinations thereof. Compared with a keyword-based similarity measure, a content-based similarity is complicated by the need for special techniques, e.g., from the area of information retrieval. However, any computation of similarity still needs to be completed within a reasonable time limit.

2.1. Document tree model

To calculate similarities among web documents, we use a model based on the document model in [7], wherein structured web documents are represented as unordered labeled trees. That is, we consider containment rather than order of appearance of words within a document. Our model differs from that in [7] in two ways: first, we don't consider the HTML formatting elements and, second, we consider a document's structure to be based on sectional elements, e.g. *Abstract* and *subsections*, while their work specifies texts in terms of pairs of start and end tags, e.g., <table> … </table>, <ul. … .

In the resultant tree, each non-leaf node corresponds to a subsection of the document (e.g. characterizing the title of the subsection), except that the root-node might also contain a set of *keywords*, a list of *authors*, a string for *title*, or/and a set of words comprising the *abstract*. Each leaf node corresponds to the text of that (sub)section. Notably, such a structure allows us to determine sectional similarities between particular elements such as titles; between the various contents, and, implicitly, between the structures of compared documents. In brief, a document tree is an unordered tree wherein each node is characterized by an associated set of type-value pairs.

Given a document tree T, of root r, with a node n_r we may represent a sub-tree of T rooted at n_r as $T(n_r)$. We define a set of functions, each characterizing some element on the document tree: *keyword*(r), *title*(r), *authors*(r), *abstract*(r) and *text*(r). For a document tree rooted at r, *keyword*(r) = $\{s \mid s$ is a keyword contained in the keyword section of $r\}$. The *title*(r), *authors*(r) and *abstract*(r) can be defined similarly. If $n_1, n_2, \ldots, n_k$ are child nodes of r, then

$$text(r)=\begin{cases} title(r) \cup \cup_{i=1}^{k} \{s \mid s \in text(n_i)\} \\ \quad \text{if } r \text{ is a non-leaf node, with children } n_1, \ldots, n_k \\ \{s \mid s \text{ is a word in leaf}(T(r))\} \quad \text{if } r \text{ is a leaf} \end{cases}$$

Essentially *text*(r) is a set of words contained in the various strings associated with nodes of the (sub-)tree rooted at r. Note that text(r) is defined recursively.

Our similarity computation algorithm works on this tree structure by exploiting the information contained in individual nodes and the whole tree. Observe that each node keeps track of its level in the tree, its content and the content of its child nodes.

2.2. Document similarity measures

Using the text extracted from elements of document (sub-)trees, we can define levels of document similarity measures. To compute the similarities efficiently, the measures must be normalized, allowing the comparison of pairs of documents and the selection of different levels of elements. Given two document trees T_1 and T_2, and two nodes $r_1 \in T_1$ and $r_2 \in T_2$, we define

$$intersect\,(w(r_1), w(r_2)) = \frac{|\, w(r_1) \cap w(r_2)\,|}{|\, w(r_1) \cup w(r_2)\,|} \qquad (1)$$

where $w(r)$ is a set of strings associated with nodes of the (sub-)tree rooted at r. The function *intersect* $(w(r_1),w(r_2))$ returns the percentage of the number of common words divided by the number of all words that appear in both $w(r_1)$ and $w(r_2)$. Clearly, *intersect* $(w(r_1),w(r_2)) \leq 1$. For two document trees rooted at r_1 and r_2, similarities of keyword, title and abstract may be defined by the following formulae (2) through (4):

$$SIM_{KB}(r_1, r_2) = intersect(keyword(r_1), keyword(r_2)) \quad (2)$$

$$SIM_{TB}(r_1, r_2) = intersect(title(r_1), title(r_2)) \quad (3)$$

$$SIM_{AB}(r_1, r_2) = intersect(abstract(r_1), abstract(r_2)) \quad (4)$$

while the content-based similarity is defined as

$$SIM_{CB}(r_1, r_2) = intersect(w(r_1), w(r_2)) \quad (5)$$

where $w(r_i) = text(r_i) \cup keywords(r_i) \cup abstract(r_i)$, $1 \leq i \leq 2$.

Let $weight_r(s)$ be the number of appearances of the word s in document represented by r, then the intersect function can be more generally defined as

$$intersect_{wt}(w(r_1), w(r_2)) = \frac{\sum_{s \in w(r_1) \cap w(r_2)} \min\{weight_{r_1}(s), weight_{r_2}(s)\}}{\frac{1}{2}\sum_{s \in w(r_1) \cup w(r_2)} |weight_{r_1}(s) + weight_{r_2}(s)|} \quad (6)$$

Based on this function, the *weighted* similarity measures $SIM_{KB}(\ )$, $SIM_{TB}(\ )$, $SIM_{AB}(\ )$ and $SIM_{CB}(\ )$ can all be defined by replacing *intersect*( ) with $intersect_{wt}(\ )$ defined in (2) to (5) above.

2.3. Data pre-processing

We developed a text filter to extract meaningful words from sections of a document, and count them per section. The method is described briefly below: In the text filter, raw text is first parsed into generalized words, called *tokens*. Tokens include meaningful strings, abbreviations, punctuation and other specialized symbols that have been derived from the structure found in the document's sections. For example, while typical words such as "web" and "page" are taken as tokens, the punctuation mark "$" and the URL "www.ecu.edu.au" are also tokens. However, digits and others insignificant words, e.g. pronouns and prepositions, are not treated as tokens.

The text filter produces a list of *(token, c(token))* pairs for each section, where *c(token)* is the count of that token within the section – in effect, a *bag-of-words* basis for our representation. Note that for brevity of the token list and subsequent comparison, each word is reduced to its stem (e.g., *server* and *service* into *serve*). While the unordered *bag-of-words* model will not suffice for linguistic analysis, we assume it captures most of the information needed for calculating similarities using formula (2) ~ (5).

3. The similarity-based web cache scheme

In this section, we describe a similarity-based multi-cache web caching scheme and on-line algorithm to capture and maintain an apposite similarity profile of documents requested through a caching proxy. The architecture consists of four major components: *central router*, *similarity profiles (SP), sub-caches,* and *document allocator*. Of these, the central router is pivotal in controlling and coordinating the other components.

Before configuring the multi-cache web caching architecture, we first cluster documents in cache based on the similarity measures introduced in (2)~(6), and determine the number of themes, N, of the documents. For each theme/cluster, a number of *stems* relating to it were chosen (e.g., by looking at all stems produced by the text filter when SP vectors were computed). Then the cache is divided into N+1 sub-caches. Each of the first N sub-caches stores documents of one particular theme, and the last sub-cache stores other documents not belonging to any of the N themes. In this way, we ensure that similarities among documents in any sub-cache are relatively higher, while relegating those among documents across sub-caches.

The SP comprises N two-dimensional arrays $A_i(*, *)$, $i=1, 2, ..., N$, of which each corresponds to one of the first N sub-caches. For each document j in sub-cache i, SP counts the number of occurrences of the stems that relate to the theme of the sub-cache, storing the numbers in vector $A_i(j,*)$. This information is useful when performing similarity-aware pre-fetching from the sub-cache to a client. For each theme, we limit the number of stems to be 100.

A sub-cache is an independent cache that has its own cache space, contents and replacement policy. Since documents in a same sub-cache are usually of similar theme, simpler replacement policies, e.g. LRU, LFU and FIFO, may be applied.

The sub-cache allocator assesses comprehensively a candidate set of evictions selected by sub-caches, with possible results of: re-caching, eviction or probation. Of these, re-caching and eviction are instantaneous, while a probation document will be held by the allocator in its own space pending a final decision.

The similarity-aware caching algorithm responds to a request for a document d as follows: an instance of d is sought in an in-cache index; if d is already cached (cache hit) and still fresh its containing sub-cache is noted whereupon d will be returned to the requesting client. If the instance of d is not fresh, then re-cache from an origin server, updating related parameters such as SP vectors. For a cache miss, the request for d will be forwarded to the origin server and a resultant

downloaded document d_{new} is returned to the client. Based on the content of d_{new}, a SP vector will be calculated to determine a sub-cache c_d in which d_{new} is to be cached. Where there is insufficient space for d_{new}, the sub cache c_d makes room according to its eviction (e.g. LRU, LFU) and/or space sharing policies. The document allocator of c_d will then assess and purge any eviction candidates.

The central router mediates between cooperating sub-caches. Although a document may be cached "conceptually" in several sub-caches in terms of sub-cache document allocator evaluation, only one actual copy will be maintained.

4. Agent-based web document pre-fetching

We focus on web document pre-fetching between caching proxies and browsing clients in this section. If the proxy can predict cached documents a user might access next, the idle periods of network links may be used to push (or to have the browser pull) them to the user while the user is viewing the web document. Since the proxy only initiates pre-fetches for documents in its caches, there is no extra internet traffic increase.

4.1. The agents

An agent is a software entity that carries out some set of operations on behalf of a client/program with some degree of independence or autonomy. An agent employs some knowledge or representation of the client's goals or desires. According to [10], agents have the following properties: (1) autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state. (2) social ability: agents interact with other agents (and possibly humans) via some kinds of agent-communication language. (3) reactivity: agents perceive their environment, and respond in a timely fashion to changes that occur in it. (4) pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative. (5) mobility: able to transport themselves from one machine to another. (6) learning: changes their behaviour based on their previous experience.

In this study, we employ both proxy-side and client-side agents that exchange messages using a predefined protocol for actualizing similarity detection, document prediction, network traffic monitoring and proxy-client coordination intentions during the process where they negotiate to reach the most probable solution. We are concerned with coordinating intelligent behaviour among these agents, i.e. how they coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems.

Three activities are crucial in the similarity-aware web document pre-fetching process and are the focuses of this section, including:

- identifying similarities between documents in the proxy cache and the document a user is viewing;
- predicting documents that a client is most likely to access next; and
- monitoring idle network periods to pre-fetch the documents.

Among these activities, the first one is similar to the similarity detection in caching new document (see Section 3). The third activity involves traffic handling and resource utilization, and is, thus, beyond the scope of this paper. Therefore, we focus on the second activity. In this architecture, agents and other software components are described as follows:

Client Agent (CA): The agent plays the role of a client. It delivers a pre-fetching request to the *Coordination Agent (CoA)*. Upon receipt of an initial pre-fetching plan (i.e., a list of candidate documents to be pre-fetched) from the CoA, it modifies the plan by removing the candidates that were hit by its local cache, and then returns the modified plan to the CoA for final pre-fetching.

Coordination Agent (CoA): the agent is responsible for receiving the pre-fetching requests from clients, and coordinates among the similarity detection agent (SDA), access pattern matching agent (PMA), pre-fetching agent (PFA) and network traffic monitoring agent (TMA) for document pre-fetching process. Through the interaction between the agents and the client in the architecture, the detailed job of the CoA involves following steps:

- receive pre-fetching requests from CAs;
- invoke SDA to identify a set of cached documents (in one or more sub-caches) whose similarities with the document the client is viewing surpass certain threshold;
- invoke PMA to assess and identify a set of users' past (historical) access patterns that could be referenced for prediction of future access of the client;
- upon receipt of the responses from steps 2) and 3), assign a process that calls PFA to produce an initial pre-fetch plan (e.g., a list of candidate documents for pre-fetching);
- send the initial pre-fetch plan to the CA to determine which in-list candidates should not be pre-fetched due to local cache hit; and
- upon receipt of the modified pre-fetch plan from a CA, assign the plan to a TMA for document pre-fetching.

Similarity detection agent (SDA): The agent determines a set of documents whose similarity with the given document surpasses the given similarity threshold. These documents will be referenced when similarity-aware PFA performs document prediction.

Access pattern matching agent (PMA): The agent matches a number of other users whose past access patterns with the given user's is greater than or equal to a certain threshold. These access patterns will also be referenced when the PFA performs document prediction.

Pre-fetching agent (PFA): The agent is responsible for predicting a set of cached documents as candidates of an initial pre-fetching plan (see Section 4.2).

Network traffic monitoring agents (TMA): the agent is responsible for monitoring the network traffic between the proxy and a given client. Once a suitable idle period is identified, the agent sends (if a proxy-side agent) a candidate document of the pre-fetch plan from the proxy cache to the client within the idle period. This monitoring-identifying-sending process continues until all candidate documents were sent, or the pre-fetching time limit is reached.

Conversation Manager (CM): The CM coordinates the activities of agents in the documents pre-fetching circle. It is responsible for receiving events from an agent, and informing other agents of messages. For example, each agent routes all its outgoing messages through the CM, and all its incoming messages are received via the CM as well.

4.2. The Agent-based pre-fetching prediction

Two agent-based algorithms are proposed to guide similarity-aware pre-fetching from proxy caches to clients. The first one is a pure similarity-based pre-fetching predictor which considers only those documents whose similarities with the document in viewing surpass a certain threshold. The second algorithm (i.e., similarity-aware pre-fetching) combines the *prediction by partial matching (PPM)* method [11] and the pure similarity-based pre-fetching strategies. These algorithms are the main functionalities and behavior of PFAs.

4.2.1 Similarity-based pre-fetching predictor. The similarity-based pre-fetching agent evaluates the next k documents in the proxy cache based on document similarities. With the support of the similarity-aware web cache architecture, the similarity-based document pre-fetching predictor works based on a very simple rule. Suppose a client is viewing a document, say d (at this time, a copy of d must be cached in a certain sub-cache, say c_i, or being held by the allocator). When a pre-fetching request is received, the CoA invokes a SDA which compute the similarities between d and those documents in sub-cache c_i by referencing the similarity information in i_{th} SP. No documents in other sub-caches are considered because of their low similarities with d. Then the predictor simply chooses k documents whose similarities with d are among the top k highest ones. These k documents, together with those cached pages to which hyperlinks exist from d, will form an initial pre-fetching plan and return to CoA for possible pre-fetching.

4.2.2. Similarity-aware pre-fetching predictor. The PPM essentially predicts the next l requests on the past m accesses of a user, limiting candidates by an access probability threshold t. The performance metrics of the algorithm depend on the (m, l, t) configurations. However, the algorithm uses patterns observed from all users' references to predict a particular user's behavior. Referencing too many contexts makes the prediction inaccurate, inefficient and unwieldy.

We extended the PPM algorithm by referencing only those access patterns from a small group of other users exhibiting high similarities in their past access patterns to predict a current user's next access [1]. The number of times the algorithm can make prediction is reduced because of the smaller sample size, but the hit ratio of the pre-fetching increases because more related access patterns are referenced. We call the method *pattern-similarity based PPM* (or *psPPM*).

We now modify *PPM* and *psPPM* by replacing the access threshold t with s, where s is a similarity threshold between the document to be pre-fetched and the document the user is viewing. Thus the new algorithm has the following parameters:

r: the number of users whose access patterns are referenced to predict future accesses of current user.

m: the number of past accesses that are used to predict future ones. We call m the *prefix depth*.

l: the number of steps that the algorithm tries to predict into the future.

s: the similarity threshold used to weed out candidate document. Only those documents whose similarity with the viewing document is greater than s, where $0 \le s \le 1$, is considered for pre-fetching.

When a pre-fetching request for document d is received from user u, the CoA invokes a PMA to assess and identify a set of r users' access patterns of relatively high similarities with u (sorted in descending order). For $l > 1$, not only the immediate next request, but the next few requests after an URL are also considered for potential pre-fetching. For example, if $l=2$, the PFA predicts both the immediate next and its successor for the user. If $m>1$, more contexts of the r

users' past accesses are referenced for the purpose of improving the accuracy of the prediction.

The PFA maintains a data structure that tracks the sequence of URLs for every user. For prediction, the past reference, the past two references, and up to the past *m* references are matched against the collection of succession to the users' past access patterns to produce a list of URLs for the next *l* steps. If a longer match sequence can be found from the other *r* users' patterns, the next URL to the longest match is also taken as a potential document to be accessed next by the user. The outcome of each prediction is a list of candidate documents, ordered by their similarities with *d*. For those candidate documents with the same similarity value, the URL matched with longer prefix is put first in the list.

5. Simulations

Two series of preliminary simulations have been conducted. The first is to demonstrate the capability of our similarity measures for document comparison to determine the document themes (or clusters). Using the obtained similarity information, our second series of simulations demonstrates the improvement in prediction accuracy (and thus hit rate) of the pre-fetching between caching proxies and browsing users using our similarity-based/aware predictors. The preliminary results indicate that our predictor is capable of practical prediction for web document pre-fetching in the sense and an improvement of the order of 10% over traditional PPM. Detailed results and their analysis are omitted here due to space limitation.

6. Conclusions

This paper proposed an agent-based similarity-aware web document pre-fetching scheme. We presented its underlying web-caching architecture and developed agent-based similarity-aware predictors for web document pre-fetching between proxy caches and browsing clients. Preliminary simulations have shown that our predictor is capable of practical prediction for web document pre-fetching in the sense that it may predict more accurately and effectively than the traditional PPM does by only referencing a reduced set of users' past access patterns.

References

[1] Xiao, J., Zhang, Y., Jia, X., and T. Li. Measuring Similarity of Interests for Clustering Web-Users. *Proceedings of the* 12th *Australian Database Conference 2001 (ADC'2001)*. Gold Coast, Australia, 107-114, 2001.

[2] Salton, G., *Automatic Information Organization and Retrieval*. McGraw-Hill, 1968.

[3] Rasmussen, E., *Clustering algorithms*. *Information Retrieval: Data Structure and Algorithms*. Prentice Hall, 419-442, 1992.

[4] Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W., and Harshman., R.A., Indexing by Latent Semantics Analysis, *Journal of the Society for Information Science*, 41(6), 391-407.

[5] Dean, J., and Henzinger, M.R., finding Related Pages in the World-Wide Web. Proceedings of the 8th International Conference on World Wide Web, 1999.

[6] Pitkow, J. and Pirolli, P., Life, Death, and Lawfulness on the Electronic Frontier. *Proceedings of the Conference on Human Factors in Computing Systems,* Atlanta, Georgia, 1997.

[7] Flesca, S. and Masciari, E. Efficient and Effective Web Change Detection, *Data & Knowledge Engineering*, Elsevier, 2003.

[8] Fox, E., Extending the Boolean and Vector Space Models on Information Retrieval with P-Norm Queries and Multiple Concepts Types. *Cornell University Dissertation*.

[9] Popescul, A., Flake, G., Lawrence, S., Ungar, L.H., and Gile, C.L., Clustering and Identifying Temporal Trends in Document Database. *Proceedings of the IEEE advances in Digital Libraries*, Washington, 2000.

[10] Bradshaw, J.M., *Software Agents*. San Francisco, CA, USA: AAAI Press/MIT Press, 1997.

[11] Fan, L., Cao, P., Lin, W. and Jacobson, Q. Web Prefetching between Low-Bandwidth Client and Proxies: Potential and Performance, *SIGMETRICS'99*, 1999.