A Semantic Model for Hypertext Data Caching

Kai Cheng[1,2], Yahiko Kambayashi[1]

[1] Department of Social Informatics
Graduate School of Informatics, Kyoto University
Sakyo Kyoto 606-8501, Japan
{chengk, yahiko}@db.soc.i.kyoto-u.ac.jp
[2] School of Computer, Wuhan University, Wuhan 430072, China

Abstract. In this paper, we propose a semantic model to capture the semantic locality in hypertext access for client–side caching. To characterize hypertext data from the perspective of clients, we define a *semantic region* as a cluster of semantically related *logical documents*. A logical document is defined as a sequence of subsequently visited interconnected *documents* which in turn are composed of a container file and (optionally) a set of component files. This model makes it easy to deal with temporal locality, spatial locality and semantic locality in hypertext access. To verify the proposed model, we use an experimental hypertext system, called HyperDB. We generate a set of workloads and assess the performance of a set of caching algorithms using the synthetic workloads and the experimental hypertext system.

1 Introduction

Hypertext and hypermedia data has recently gained importance due to the success of the world-wide web and hypertext transfer protocol (HTTP). A salient feature of a hypertext system is that users search for desired data primarily by navigation or following links from one document to another, while query facilities, if any, are just used for pruning the navigation space [7]. The interactive nature of hypertext navigation poses challenges in system performance, especially in a client/server architecture where downloading a document from remote servers is much more time-consuming than reading from a local disk.

Caching has been widely used to alleviate performance bottlenecks in computer architecture, network and database systems[19]. Particularly, caching that utilizes computational and storage resources of the client machines has been a key solution to achieve high performance and scalability in client/server database systems [9]. Semantic caching is an advanced form of client caching suited for utilizing client resources, which maintains a semantic description for cached data, called *semantic region*, to capture the *semantic locality* of data usage [8]. Semantic caching gives higher priority to data relevant to the frequently used data as it believes those semantically related data are more likely to be accessed again in the near future.

In a query–retrieval based database system, semantic description of cached data can be straightly obtained from the formulas of query constraint. Taking as

an example from [8], suppose we want to find all employees whose salary exceeds 50,00 and whose age are at most 30 years old. We can issue a query using this constraint formula $Q_1 = (Salary > 50,000 \wedge Age \leq 30)$. For caching of data with explicit semantic description, one can easily make use of query results to serve the subsequent query requests according to their semantic containment: only the remainder of a query should be issued to get complimentary data.

In a navigation–based hypertext system, however, there is no explicit semantic description altogether, instead the user expresses his information needs by interacting with the system, i.e. repeatedly choosing new data while looking at current data items [7]. As a result, caching of hypertext data cannot utilize the explicit semantic description to capture semantic locality. Another problem for caching hypertext data is that hypertext documents are inter–connected by hyperlinks and each document is also composed by a set of small components of other media. To take advantage of the features, a suitable model is essential.

Modeling hypertext data has been well researched in hypertext community [2, 5, 10, 11, 13, 23]. Dexter model [11] identifies the relevant abstractions found in a wide range of existing hypertext systems, providing a common vocabulary and its meaning in order to talk about hypertext systems. The Fundamental Open Hypertext Model (FOHM) [13] expands the Open Hypermedia Protocol (OHP) data model to describe a broader set of hypermedia "domains", such as navigational domain, spatial domain and taxonomic domain, to meet the requirements of interoperability between hypertext systems. The OHP protocol was always more concerned with navigational hypertext, whereas FOHM is capable of expressing all three domains.

Hypertext abstract machine (HAM) [6] is an architectural description of a general-purpose, transaction-based, multi-user server for a hypertext storage system. Furuta and Stotts' Trellis model [21] is a formal specification of hypertext based on petri nets. Hypertext has also been formalized as graphs [22] or automata as in [14, 17], where the authors studied the dynamic properties of hypertext in terms of "reader's experience", formalizing what readers see when they interact with a hypertext system. *Web machine* [1, 12] or *web automata* [20] are computation models for querying the web, which paid much attention to the navigational nature of the web.

The afore–mentioned models, however, either aimed at facilitating authoring activity and system design, or attempted to investigate computation mechanism for designing suitable query languages. To capture temporal locality, spatial locality as well as semantic locality, we need a model that can handle structural organization, browsing semantics as well as content relevance from the perspective of the user. As a client cache can only see a restricted fraction of the whole hypertext system and as there is tradeoff between model complexity and associated overhead, current work is not suitable for the caching purpose.

In this paper, we propose a new semantic model to capture the semantic locality of hypertext data caching. We define a *semantic region* in a cache as a cluster of semantically related *logical documents*. A logical document is defined as a sequence of subsequently visited interlinked *documents*, which in turn are

composed of a container file and (optionally) a set of component files. We verify the proposed model by implementing an experimental hypertext database, called HyperDB. We then generate a set of workloads and assess the performance of a set of caching algorithms using the synthesized workloads and the simulated hypertext system.

The rest of paper is organized as follows. Section 2 proposes the semantic model. Section 3 describes a virtual hypertext database based on the proposed model, called HyperDB as well as workloads with various kinds of locality of reference for HyperDB. Section 4 gives a set of caching replacement algorithms that take into account various forms of reference locality, describing simulation results obtained under the synthetic workloads and the simulated hypertext system. Section 5 concludes the paper and describes some future directions.

2 Modeling Cached Hypertext Data

Hypertext data is a collection of documents (or "nodes") containing cross-references or "links" which, with the aid of an interactive *browser* program, allow the reader to move easily from one document to another. The extension of hypertext to include other media – sound, graphics, and video – has been termed "hypermedia", but is usually just called "hypertext", especially since the advent of the World-Wide Web and HTML.

In this section, we develop a semantic model by taking into account both the structural as well as the semantic features of hypertext data. Particularly, we define the concept of semantic region for hypertext data caches. This model characterizes a collection of data in a hypertext cache from three abstraction levels: physical documents, logical documents, and semantic regions based on physical structure, logical structure and semantic structure respectively.

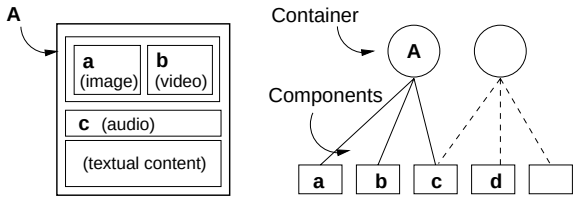

Fig. 1. Document Composed by Hypermedia Components

2.1 Physical Structure of Hypertext Data

To capture the spatial locality of hypertext access, a cache should first understand physical structure of hypertext documents, the basic elements in a hypertext system. We describe this feature mainly following the Dexter model [11].

First, *document* is a basic element of a hypertext system. We define a (hypertext) document as a composition of a container (file) and (optionally) a set of media component files that represent media other than text such as image, audio and video (Fig. 1). A container (file) consists of (1) textual content, a sequence of terms, sentences, paragraphs; (2) anchors and (3) hold places for media components.

An *anchor* is a point in a document representing a start point for a link. An anchor also specifies a valid range or *anchor text*, indicating what part of a document belongs to the anchor. Anchor texts often describe the linked document, used as a navigation guide to the information the user is seeking for. The anchor text of a is denoted by $text(a)$. Anchoring provides a mechanism for addressing locations within the content of a document.

Link is another basic element in a hypertext system. A *link* represents relations between documents. There are two kinds of links. A link from one anchor to another anchor is called span-to-span link, while a link from one anchor to a document is called as span-to-node link (Fig. 2). In the following, we only consider span-to-node link, and represent a link as a triplet $< d_1, a, d_2 >$, where a is an source anchor in document d_1, d_2 is a destination document of this link.

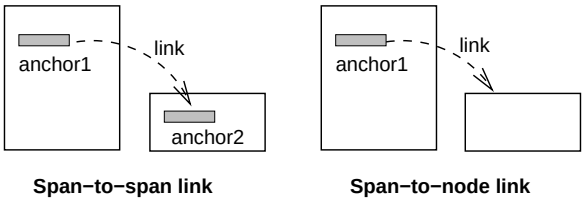

Fig. 2. Two Types of Link

Documents can be evaluated in terms of size, recency and frequency of reference. To measure the relevance of a document to some interested topics (later we call "semantic regions"), textual content (terms or sentences) will be evaluated on the basis of techniques in information retrieval (IR), such as vector space model (VSM) and TF–IDF scoring scheme. The content of a document d can be expressed as

$$contnentof(d) =< title, body >$$

where *title* is a sentence that describes the content of the document, and *body* is a sequence of terms in the document. Media component (files) are embedded in the hold places of container files. A media component file can be shared by one or more documents, thus whether a component file can be deleted by a garbage collector is determined by not only how often it has been used as in

most caching schemes, but also determined by whether there is no more used by existing cached documents.

A hypertext database is often modeled as a directed hypergraph, with documents as nodes and links as edges. This model however is not suitable for client caches because a client cache does not see the whole structure of the potential hypergraph, instead what it can see are paths followed by the user in that hypergraph. To predict how the user uses the hypertext database for caching decision making, we should model the paths that the user often traverses, instead of the whole hypergraph.

2.2 Logical Documents: Logical Structure of Hypertext Data

As links created by hypertext authors do not always reflect what readers think, a cache often sees a subset of documents and a small fraction of paths (sequences of document-links) are often traversed. In a navigational access environment, users are apt to travel data items back and forth in accordance with paths. Thus, data items might be visited just because of it location, rather than its content. We define a path frequently traversed by some users as a *logical document*.

A logical document is a representation of user's perspective of the hypertext data. In other words, how authors created a hypertext database is one thing, while what the client would be interested is another. This distinguishes our model from any other hypertext models created from the point of view of hypertext authors or system designers.

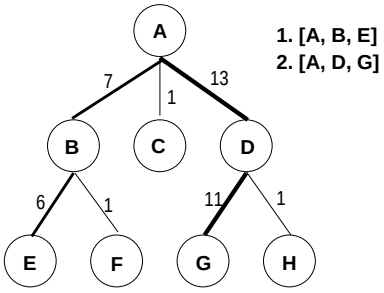

Fig. 3. Logical Document Based on Repeating Traversal Paths

Fig. 3 depicts two logical documents in a hypertext database: one is "A–B–E", the other is "A–D–G". In "A—D–G", starting from document **A**, the user often (13 times) chooses to follow a link to **D**, then **G**. It is reasonable to think that, for the user of the cache, "A–D–G" is a logical unit that contains specific information he needs. The first document in the path of a logical document is called an "entry document", while the last document traversed in the path is called a "terminal document".

Logical documents can be measured in terms of size, recency, frequency of reference. The size of a logical document is the length of path, which is indeed the number of documents contained in the path. A reference to a logical document is defined as a successful traversal starting from the entry document, walking through a link to the second document on the path within a limited time interval, and so on, until reaching the terminal document.

Logical documents represent the readers' viewpoint of hypertext data. That is, different paths leading to a same document imply different perspectives of the user, To deal with this difference, we define the content of a logical document to be $< title, body >$ with $title$ being the union of anchor texts contained in the path and the title of the terminal document. As shown in Fig. 4, suppose we

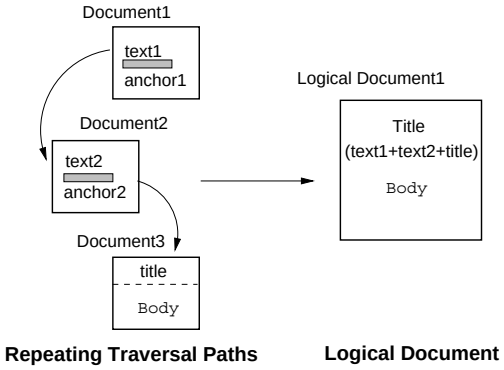

Fig. 4. Link Navigations Imply Semantic Specificity

have a logical document $l =< [d_1, a_1], [d_2, a_2], [d_3] >$ where d_i $(i = 1, 2, 3)$ are documents in the repeating traversal path, a_i $(i = 1, 2)$ are anchors leading to a subsequent document. That is, the user first follows a link from anchor a_1 in d_1 to d_2, where he follows another link from anchor a_2 to d_3. Let $text(a_i)$ be the anchor text of a_i, $tile(d_i)$ and $body(d_i)$ be the title and body of a document respectively. Then we can define the content of logical document L to be

$$contnentof(l_i) < text(a_1) + text(a_2) + title(d_3), \quad body(d_3) >$$

Here "+" is string concatenation operation as in a typical programming language. For example, if the anchor texts on the path of a logical document are "Travel in Kyoto", "List of bus stations" and "Kyoto station", and the title of the terminal document is "Access to the Sinkansen superexpress", the the logical document will have a logical title "Travel to Kyoto, List of bus stations, Kyoto station, Access to the Sinkansen superexpress".

Note that logical documents can be of different sizes depending on the configuration of implementation and the actual usage status. A special case is when

the size is 1, which means there is only one document included in the logical document. Thus, each visited document can a logical document.

2.3 Semantic Region: Semantic Structure of Hypertext Data

Semantic regions is a concrete description about user interests, which play an important role in identifying preference of users. We denote a semantic region as $R = (\sigma, \lambda)$, where σ is the semantic centroid (cluster center, or median). λ is the radius of the semantic region. A semantic region is a cluster of logical documents with a semantic centroid such that each logical document belongs to exactly one most suitable cluster, that is, it is closer to centroid of this cluster than any others. The centroid of a cluster is represented using a feature vector based on vector space model (VSM) and TF–IDF scoring scheme. For example, (30, 34, 120, 10) is a feature vector presentation with respect to (bread, butter, salt, knife).

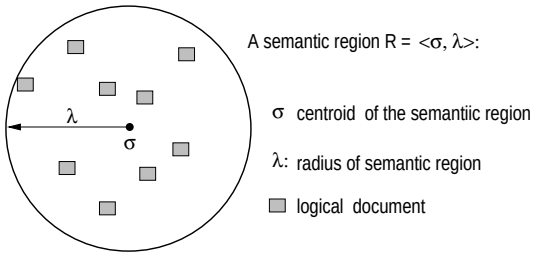

Fig. 5. Semantic Region Based on Adaptive Clustering of Logical Documents

As new documents come continuously, determining semantic regions for hypertext caching requires efficient single–pass clustering algorithms that consume a small amount of memory. Fortunately, there exist a number of streaming–data algorithms that can achieve high quality clustering [24, 4, 15]. In general, a clustering problem cane be described as follows: given the number k of clusters, a clustering algorithm will try to find k centroids so that each data point is assigned to the cluster defined by the centroid nearest to it. This is also known as "k-Median" problem.

Suppose the quality of clustering is measured by the sum of square distance of data points from their centroid, the randomized algorithm $LSEARCH$[15] can usually find a near–optimum solution in $O(nm + nk \log k)$ of time, where n is proportional to the number of data points, m is a small number. In this work, we will not evaluate various clustering algorithms, instead we assume we already know a suitable near–optimum algorithm that can always cluster new logical documents received. We will concentrate on exploring whether higher–level semantic information can help determine potential usage of hypertext data.

As content of a logical document has two parts: title and body, we need a method to combine them together. As terms in a title are generally more important than those in a body, we show stress more on title than on body. Suppose $\mathcal{L}$ is the set of logical documents for a hypertext cache. l_i is a logical document with content $< title, body >$. Let v_i^{title} and v_i^{body} be the TF–IDF based feature vectors for title and body of l_i respectively. The comprehensive feature vector of l_i can be calculated as a weighted sum of both, that is,

$$v_i = \omega \cdot v_i^{title} + v_i^{body}$$

Here ω is a parameter larger than 1. The combination of feature vectors for title part and body part enable to distinguish two logical documents even when they have the same terminal document. Again we consider the example about "how to access to Sinkansen superexpress" in "Kyoto station". Another reader may reference the same document after following a list of "NTT Western Japan", "Kyoto Office", "Location" , and then the terminal document. The first logical document is likely for general travelers, while the second logical document is much more suitable for business travelers.

3 HyperDB, Benchmarking Hypertext Semantic Caching

To verify the proposed model and to evaluate caching schemes with consideration of temporal, spatial, and semantic locality, we set up an experimental hypertext database system as a testbed. However, as mentioned before, currently we concentrate on evaluate semantic information derived according to our semantic model can be a help in better cache management.

3.1 HyperDB, A Virtual Hypertext Database

We need not actually generate all these elements and not need to employ a hypertext database management system to maintain it. We simplify the proposed model because we need only to maintain a set of features that provide necessary data for calculating logical documents, semantic regions. First, we can put aside the real content of documents or logical documents, because dealing with this feature we have to do much engineering work that has done by most IR researchers. Then all calculation involved in content of documents can be omitted for the time being.

We begin with generating a set of component files, each of which has a different size and media type. We then generate a set of container files, which has a size and a corresponding set of component files. In future, we will also consider terms and their TF-IDF scores with respect to a container file, but for the time being, we need not such a complicated implementation.

1. a set of components $Components$, each element is a triplet of id, media $type$ and $size$.

2. a set of documents *Documents*, each of them has an *id*, a set of component id's and a set of terms with their corresponding TF-IDF *socres*;
3. a set of logical documents *LogicalDocuments*, with a *id*, a sequence of linked documents id's;
4. a set of semantic regions *SemanticRegions*, with a semantic centroid (a vector of weighted terms), a threshold. ;

A logical document is generated by choosing a sequence of documents (container files indeed). Each sequence has a length of 1, 2, 3 etc, indicating there is/are one or more documents.

3.2 Generating Accesses to HyperDB

For the hypertext database, we synthesize workloads. We use the following method to guarantee *temporal locality* in request stream. To generate a new request, we pick a random number and take different actions depending on the random number. If the number is higher than a certain constant μ, a new request is issued. If the number is lower than μ, we re-issue a request issued before. Thus, μ is the inherent hit ratio in the request stream.

Workload	Property	Description
Workload A	Temporal Locality	Revisit Recently Used Basic Objects
Workload B	Spatial Locality	Consider Document Composition
Workload C	Spatial Locality	Consider Link Structure
Workload D	Semantic Locality	Be Aware of Semantic Regions

Table 1. Properties for Synthesized Workloads

If we need to re-issue an old request, we choose the request issued t requests ago with probability proportional to $1/t$. To determine this t, we maintain a sum S of $1/t$ for t from 1 to the number n of requests issued, that is $S = \sum_{t=1}^{t=n} 1/t$. Every time it is necessary to issue an old request, we pick a random number from 0 to 1 (call it r), calculating $r \cdot S$, and chooses t where,

$$\sum_{i=1}^{i=t-1} 1/i < r \cdot S < \sum_{i=1}^{i=1} 1/i$$

In essence, t is chosen with probability $1/(S \cdot t)$ and the recently issued requests are more likely to be re–issued. Temporal locality combined with hypertext structures and semantic regions, derives other kinds of reference locality, i.e. *spatial locality* and *semantic locality*, where spatially related or semantically related data items are more likely to be re–requested in a short period of time.

In the following, we apply this method in maintaining temporal locality at component–level, document–level, path-level and semantics-level, obtaining four groups of request streams. Table 1 lists the primary profiles of these workloads.

The workload A is generated on the basis of reference locality for basic objects without considering structural or semantic locality. Workload B is based on reference locality for documents, thus media components are accessed only when their container files are accessed for the first hand. Workload C is further based on logical documents. The recently accessed logical document will be more likely be re-accessed in the near future due to the temporal locality for logical documents. When a logical document is accessed, all member documents will with very high probability be accessed and all component files of those documents will be accessed consequently

Finally, workload D is based on reference locality of semantic regions. When a semantic region has recently been referenced (one of its member logical document was referred), then similar logical documents in the same semantic region are more likely to be reused in the near future. Within each semantic region, the recently used logical documents are more likely to be reused. We call the temporal locality for semantic regions as well as logical documents as "multiple temporal locality".

4 Semantic Caching for HyperDB

Based on the semantic model developed so far, we can now design algorithms for cache management. The baseline algorithm is LRU-K, proposed by E. J. O'Neil et al in [16]. The basic idea of LRU–K is to keep track of ΔT_K, i.e. time since last K'th reference (or infinitely large if there are no more than K references), using this information to estimate the popularity of a data item. The $K/\Delta T_K$ is usually called "dynamic access frequency" of a data item. A cache algorithm will try to find the least frequently used data (with smaller access frequency) and replace it with more popular data (with larger access frequency).

4.1 Size-Adjusted LRU-K (LRU-K-S)

As basic objects in hypertext databases are not identical in sizes, we should extend LRU-K to deal with the heterogeneous sizes of hypertext objects. The idea is to normalize miss penalty by data size, $K/(\Delta T_K \cdot Size)$, then use the normalized cost function to measure the potential of a data item. Consequently, we obtain a Size-Adjusted LRU-K (call it LRU-K-S).

4.2 Structure-Aware Caching (LRU-K-T)

Hypertext data has some structure where component data depend on container: when a container data item was accessed, all its component data will automatically be accessed. Conversely, when a container data item is replaced from a

cache, the related components should also be deleted except some are shared by other containers.

So far, this kind of structure–based dependency has not been well addressed. We extend LRU-K to incorporate this property, assuming that each component file maintains a *reference counter*, indicating how many documents are currently sharing this component. Before the counter becomes zero, cache priority of this component depends on its container. When all its containers are removed (counter becomes zero), the component file will be deleted, similar to the idea of "garbage collection" in memory management. In this way, we obtain a Structure-Aware LRU-K (call it LRU-K-T).

Algorithm 1 LRU-K-C Content-Sensitive LRU-K

Require: $space$ = unused cache space, Ω is a set of semantic regions

```
1: for each request q for data item, d_0 do
2:   if d_0 is in cache then
3:     return a copy of d_0
4:   else
5:     Retrieve d_0 from database;
6:     P_0 = getEmbedders(d_0);
7:     T_0 = ⋃_{p∈P_0} getSemanticRegion(p);
8:     t_0 = getLeastSemanticRegion(Ω);
9:     t = getMostSemanticRegion(T_0);
10:    if isPriori(t, t_0) then
11:      P = getDocuments(t);
12:      while size(d_0) > space and ¬empty(P) do
13:        p_0 = getLeastDocument(P);
14:        D = getEmbedded(p_0);
15:        while size(d_0) > space
16:        and ¬empty(D) do
17:          d = getLeastObject(D);
18:          removeObject(d, D);
19:          space+ = size(d)
20:        end while
21:        if empty(D) then
22:          removeDocument(p_0, P);
23:          updateCentroid(t)
24:        end if
25:      end while
26:      LRU-K-S(p_0, d_0);
27:    end if
28:  end if
29: end for
```

4.3 Popular Path-Aware Caching (LRU-K-P)

Another extension to the standard LRU-K is making use of frequently traversed paths, namely, logical document. Recall that logical document is a larger granularity for cache management. As we only care about links that has been used at least once, we should not maintain a large collection of links. From the user's perspective, a never–used link may be of little interest and will less likely to be visited in the near future.

The times of last K traversals are recorded so that the "dynamic access frequency" about this logical document can be calculated. Next time when the user begin to visit a document at this path, cache manager will give higher priority to those documents on the paths or logical documents that have been "most recently and most frequently used". This is a Popular Path-Aware LRU-K, called LRU-K-P.

4.4 Content-Sensitive Caching (LRU-K-C)

Finally, we reach the point to develop the semantic caching scheme. The semantic region for navigational access is built upon document clustering techniques. First let us define some functions for managing priori ty queues on semantic regions, logical documents, documents and basic objects (single physical objects/files including both containers and components).

- $getEmbedders(d)$ returns a all documents that embed the object d
- $getEmbedded(p)$ returns a all objects embedded in the document p
- $getSemanticRegions(p)$ returns all semantic regions that a document p belongs to
- $getDocuments(t)$ returns all documents in semantic region t
- $getMostSemanticRegion(T)$ is a function for selecting a semantic region with highest priority in a set T of semantic regions
- $getLeastSemanticRegions(T)$ is a function for selecting a semantic region with lowest priority in a set T of semantic regions
- $getLeastDocument(L)$ returns a document with LEAST priority in set P
- $getLeastObject(D)$, returns a object with LEAST priority in set D.

4.5 Experimental Evaluation

We assess the performance of the proposed algorithms using synthetic workloads obtained in Section 3. Although most studies on caching algorithms use trace–driven experiments or event–driven experiments [3], we choose in this work to use event–driven approach for two reasons. First, to the best of our knowledge there are no suitable benchmarks available for our purpose which include both a collection of hypertext data and the workloads with respective to those data. Second, our algorithms, especially content–sensitive caching, rely on techniques in other research fields, such as usage mining, text clustering etc. a thorough evaluation of all specific techniques is difficult and we will address in future work.

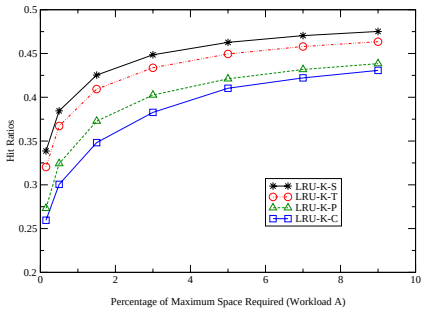

Fig. 6. Hit Ratios under Workload A

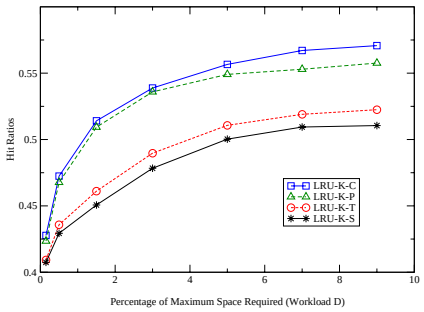

Fig. 7. Hit Ratios under Workload D

Fig. 6 shows results (hit ratios) of experiments under workload A, where no spatial or semantic locality is considered. The plots show LRU-K-S is better while others including LRU-K-C performed not so good. The reason is that the structure–aware or semantic-aware algorithms try to bias towards documents closer spatially or semantically to recently used ones, however, workload A lacks of such characteristics. This can also explain the results in Fig. 7 under workload D, where LRU-K-C performed better than others, since in workload D, logical documents within a same semantic region tends to be referenced at the same time.

5 Conclusion and Future Work

Semantic locality is the most important feature of data access, which has been exploited in semantic caching schemes for traditional query–retrieval based database systems. However, for a navigation based hypertext database system, such as the web, the advanced locality of reference has not yet been incorporated as there is no explicit description of semantic regions in the form of access. In this paper, we have proposed a semantic model that took into account both structural feature as well as semantic feature of hypertext data. Our model is especially suitable for navigational access to seeking desired data in a hyperlinked information space.

We verified the proposed model by creating an experimental hypertext database called HyperDB, then generated workloads with different locality of reference. These workloads can be used for analysis of semantic caching schemes for hypertext data. Preliminary experiments have done to evaluate some semantic caching schemes.

Our future work is to include textual content and test for efficiency under a verity of clustering schemes and parameterizing the process for constructing semantic regions.

Acknowledgments

The authors would like to thank Mukesh Mohania and Yanchun Zhang for many interesting discussions.

References

1. Serge Abiteboul and Victor Vianu. Queries and Computation on the Web. In *Proceedings of 6th International Conference on Database Theory (ICDT'97)*, pages 262–275, January 8-10, Delphi, Greece, 1997.
2. Foto N. Afrati and Constantinos D. Koutras. A Hypertext Model Supporting Query Mechanisms. In *Proceedings of European Conference on Hypertext*, pages 52–66, 1990.
3. Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the World Wide Web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):95–106, 1999.
4. Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling Clustering Algorithms to Large Databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 9–15, New York City, New York, USA, August 1998. AAAI Press.
5. Andrea Caloini. Matching Hypertext Models to Hypertext Systems: A Compilative Approach. In *Proceedings of European Conference on Hypertext*, pages 91–101, 1992.
6. Brad Campbell and Joseph M. Goodman. HAM: A General Purpose Hypertext Abstraction Machine. *Communications of the ACM*, 31(7):856–867, July 1988.
7. Chris Clifton and Hector Garcia-Molina. Indexing in a Hypertext Database. In *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, pages 36–49, Brisbane, Queensland, Australia, August 1990. Morgan Kaufmann.
8. Shaul Dar, Michael Franklin, Bjorn Jonsson, Divesh Srivastava, and Michael Tan. Semantic Data Caching and Replacement. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, Bombay, India, September 1996. `http://www.cs.umd.edu/projects/dimsum/papers/semanticcaching.ps.gz`.
9. Michael J. Franklin. *Client Data Caching*. Kluwer Academic Press, Boston, 1996.
10. Richard Furuta and P. David Stotts. A Functional Meta-Structure for Hypertext Models and Systems. *Electronic Publishing*, 3(4):179–205, 1990.
11. Frank G. Halasz and Mayer D. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.
12. Alberto O. Mendelzon and Tova Milo. Formal Models of Web Queries. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems(PODS)*, pages 134–143, Tucson, Arizona, 1997.
13. David E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: a Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains. In *Hypertext*, pages 93–102, 2000.
14. Luc Moreau and Wendy Hall. On the Expressiveness of Links in Hypertext Systems. *The Computer Journal*, 41(7):459–473, 1998.
15. Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-Data Algorithms For High-Quality Clustering. In *International Conference on Data Engineering (ICDE)*, 2002.

16. Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 297–306, New York, 1993.
17. Seongbin Park. Structural Properties of Hypertext. In *UK Conference on Hypertext*, pages 180–187, 1998.
18. Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieva l. In Karen Sparck Jones and Peter Willett, editors, *Readins in Information Retrieval*, pages 323–328. Morgan Kaufmann, 1997.
19. Curt Schimmel. *Unix Systems for Modern Architectures*. Addison-Wesley, 1994.
20. Marc Spielmann, Jerzy Tyszkiewicz, and Jan Van den Bussche. Distributed Computation of Web Queries Using Automata. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems(PODS)*, pages 97–108, Madison, Wisconsin, USA, 2002.
21. P. David Stotts and Richard Furuta. Programmable Browsing Semantics in Trellis. In *Hypertext*, pages 27–42, New York, 1989.
22. Frank WM. Tompa. A Data Model for Fexible Hypertext Database Systems. *ACM Transations of Information Systems*, 7(1):85–100, 1989.
23. Marcelo Augusto Santos Turine, Maria Cristina Ferreira de Oliveira, and Paulo Cesar Masiero. A Navigation-Oriented Hypertext Model Based on Statecharts. In *Hypertext*, pages 102–111, 1997.
24. Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *SIGMOD Conference*, pages 103–114, 1996.