Capacity Bound-free Web Warehouse

Yahiko Kambayashi, Kai Cheng

Graduate School of Informatics, Kyoto University
Sakyo Yoshida Homachi, Kyoto 606-8501, JAPAN
{yahiko,chengk}@db.soc.i.kyoto-u.ac.jp

Abstract

Web cache technologies have been developed as an extension of CPU cache, by modifying LRU (Least Recently Used) algorithms. Actually in web cache systems, we can use disks and tertiary storages since access time of disks (or even online tapes) is still shorter than time required for retrieving web pages from origin severs. Thus, we can remove the restriction of cache size that has been the most severe condition for designing cache algorithms. We still need to determine the priority of data for efficient processing. In this paper, the concept of Capacity Bound-free Web Warehouse (CBFWW) will be introduced. There are the following assumptions in conventional web cache systems. (1) Priority of each object corresponding to web contents is simply determined by using queue. (2) Each object is independent. (3) Transparency of cache is assumed, where a user cannot know the contents.

As the communication speed of the web is very slow, we can use complicated algorithms to determine the priority. Priority of newly retrieved documents is determined by computing their similarities to the documents with known priorities. Although objects with high priority are usually stored in fast access storage, we will discuss how to handle large objects. In conventional database systems, usage information like priority is hidden to the users. As in web-based applications, only a small fraction of data becomes hot spot and hot spots are changing very rapidly, self-organizing property using dynamically changing priority is important. As web data are very much complicated, we have to handle mutually related data objects. Links and structures of documents are also factors to determine the priority. As a large amount of data is stored, we should use the contents like database systems, not like conventional cache systems. Therefore transparency is not assumed. We need mining functions to analyze the usage patterns. We can retrieve objects with usage data which were not possible by cache or database system. The whole systems should have functions of cache, databases and data warehouses. We can further add useful functions.

Proceedings of the 2003 CIDR Conference

1 Introduction

This work is motivated by the following observation: with the significant advances in storage technology, storage capacity is currently not an important limiting factor for disk-resident caches such as web caches. Under such environment, we will discuss the following two problems: (1) How to organize a web cache with a little restriction of the capacity bound. (2) What kinds of new functions should be added for such advanced web caches.

In the past decade, the explosive growth of internet and world-wide web poses new challenges upon the network and database technologies. Due primarily to the popularity of the web, internet traffic has been doubling every 6 months [11], whereas according to [7] the bandwidth of the backbone only increases by 50% every year. The widening gap between the amount of the traffic and the network bandwidth results in increase of user response times.

There are mainly the following approaches for reducing the web traffic. (1) Data compression: It will contribute to reducing the amount of data to be transmitted. (2) Index organization: A good index will help to reduce the cost of unnecessary retrieval. (3) Data duplication: If there are copies available at close sites, communication cost can be reduced. For (1), there are a lot of standards and it is rather difficult to improve compression algorithms drastically. For (2), robots will search through internet and they will cause the increase of traffic. There is a trade-off problem between the quality of the index and robot usage.

For (3), mirror sites and cache are representative techniques. As one of the advantages of caching is its self-organizing capability, i.e. contents are selected automatically, it has been a common technique widely used to en-

hance internet infrastructure. Cache can be deployed at a variety of situations. Cache for content delivery networks (CDNs) is used for materialized view, which will reduce workload of origin web servers.

Traditionally, caches have assumed to have a limited storage capacity and cache replacement algorithms are dedicated to handling the situation when no space left for holding new data. However, throughout last decades, storage technologies have advanced significantly. Currently, the price of magnetic disks is as low as 700 MBytes a dollar (`http://www.computersupersale.com/`) in contrast with 24 MBytes a dollar in 1998 [9]. Since access time of disks (or even online tapes) is still shorter than time required for retrieving web pages from origin severs, we can use disks and tertiary storages in web cache systems. Thus, we can remove the restriction of cache size that has been the most severe condition for designing cache algorithms. It is not unrealistic to employ caches of Terabytes or even Petaytes to enhance the internet infrastructure in the near future.

The concept of cache originally came from cache for CPU, thus there are the following assumptions:

1. Capacity limit. Cache size is usually very small due to the limit of chip size.

2. Simple algorithm. In order not to incur too much overhead, simple algorithms are used for cache management. Sometimes special hardware support is required.

3. Transparency. Neither the cache manager nor the processor is required to investigate the contents of the data blocks in cache.

4. Performance measurement. Evaluation of cache algorithms is primarily based on hit ratio, that is, how many requests out of all are satisfied by a cache.

When caches are extended to other fields, such as virtual memory management, disk I/O or database buffering, mobile or web caching, the basic assumptions described above are not changed so much, although some minor changes have made in measurement of cache performance. For instance, to handle web data whose sizes are not equal, hit ratio is modified to byte hit ratio (hit ratio weighted by sizes of data).

Transparency is only effective for a small amount of data. When a large amount storage space is available, cache will become a giant repository of information. Transparency is not suitable, since users cannot make use of contents in the cache actively. Transparency results in great loss of cache utility. By our experiments, we found the following fact by analyzing web access logs of our research group for one month:

> *Over 60% of web pages once used will never be retrieved again before modified or replaced.*

Similar results can also be found in other studies [3, 10]. Even we employ an optimal algorithm for cache management, the majority of web data are always unused.

To deal with these issues, we have to "forget definitions and constraints used for traditional cache and to start to define them from the scratch". We can also add new useful functions. The basic assumptions are,

1. There is conceptually no limit of storage space.

2. Store everything as long as it seems to be worthwhile.

3. Users can query the contents of cache and interact with cache in a non-transparent manner.

4. Operational data (logs) are also stored for priority management and performance improvement. These data can also be used for recommendation functions.

Although we do not assume the limit of storage space, we still need to consider priority of data mainly due to the performance reasons. Besides the advantages of database functions, self-organizing capability of cache is important. We will discuss how to determine priority under web environment. The following problems are newly discussed in this paper.

First, in the traditional cache, all the data in the cache can be retrieved by similar access time, priority of data in the cache is only necessary when there is no space for newly added data. Thus it is required to determine the data with the least priority only. If a data object is not used for a long time, the priority will go down. In our system, there is a storage hierarchy and it will be a waste of main memory if newly added objects get the highest priority as 60% of data will not be reused. We have developed an algorithm to determine the priority of an object when it is retrieved, using the content relevancy (use of topic sensor, to be discussed later) and the similarity between objects which have been assigned their priority. For example, a newly added object is similar to the object with low priority, the same priority may be assigned to the new object.

Secondly, since web data are complicated, we have to develop methods to assign priorities to such composite objects. Besides links, semantic constraints will also determine the priorities of mutually related objects. For example, if object A is shared by to composite objects B and C, the number of retrievals of object A becomes bigger than those of B and C. By traditional way, object A will have high priority. In our algorithm, the priority of A will not exceed the priorities of B and C. Our definition assigns natural priority for object A.

We have discussed how to map objects with priority to appropriate storages.

1. Storage media. Where to store (main memory, disks, tapes) is determined by the priority.

2. Index structure. Detailed index is given to important documents. Some important indexes are stored in the main memory.

	Database Systems	Data Stream Systems	Traditional Data Caches
Objectives	Data Management	Online Decision Support	Efficiency
Data Store	Persistent Store	Little or No Store	Temporary Store
Storage Capacity	No Limit Assumed	Limited Memory	Limited Storage
Data Manipulation	Insert, Delete, Update	Append-Only	Insert, Delete
Query Capability	Select, Join, Project, Aggregate	(Approximate) Aggregate	Not Allowed
Management System	DBMS	DSMS [1]	Ad hoc

Table 1: Comparison among Databases, Data Streams and Traditional Data Caches

3. Levels of details. Since there may be important but large documents, we may not be able to store the whole data in the main memory, abstracted contents are prepared to be stored in the main memory in order to save space.

Under this background, in this paper, we propose *Capacity Bound-free Web Warehouse (CBFWW)*, a sophisticated manager for caches with a very little storage limitation. In addition to the functions of a conventional cache, a CBFWW implements a set of new functions as listed below. The architecture is shown in Figure 1, which will be discussed in Section 3

1. Improving utilization of data that could have been ignored by transparent accesses

2. Determining priority of data when retrieved for the first time based on popularity of similar contents. Priority is used for determining the object location in self-organizing storage structure.

3. Use of data in CBFWW as a sample of the whole web for popularity-aware search and cache-conscious navigation.

4. Detection of hot topics from news sites using a *topic sensor* to improve accuracy of priority prediction and realization of prefetching operations.

The remainder of this paper will be organized as follows. In the next section, we will review techniques and proposals related to our work. Section 3 gives an overview of a CBFWW system. Section 4 describes priority-based advanced queries and storage management scheme. To facilitate handling web data, Section 5 gives an object hierarchy model for hypertext-based data. Section 6 concludes the paper and describes some future directions.

2 Related Work

Our proposal is based on an analysis of features in databases, data stream and traditional data cache. In this section, we briefly review previous work relevant to our work, particularly we put emphasis on systems and proposals on data management in non-traditional environments.

2.1 Databases and Data Streams

Data stream refers to fast arriving data tuples, which has recently attracted strong attention [1]. Such systems become available due to the reduction of storage cost. Although data stream systems and CBFWW are handling a large amount of data, the characteristics are completely different as shown in Table 1. The major objective of a data stream model is for online decision support. Persistent store is not required and timely response is important. A data stream can be seen as a append-only table. Queries to a data stream are limited to aggregate, for instance, Max, Min, Average, mostly in an approximate way. Individual data is not so important as in traditional database systems. Data will either be discarded or archived so that it is quite expensive to retrieve old data once processed. Data Stream Management Systems (DSMSs) proposed in [1] are expected to play a similar role with DBMSs in traditional database systems.

Traditional data cache also deals with fast arriving data, although there is a strict storage limit. The objective of cache is to improve performance of systems. Although data can be inserted into and deleted from a cache, queries to a cache is generally not supported. A cache should not only handle data streams to determine reference patterns, but also should have functions to manage data storage for future use.

2.2 Mid-Tier Data Management

As the multi-tier client/server architecture becomes increasingly common in the web, *mid-tier data management*, i.e. cache with database management capabilities has recently gained importance [5, 13]. In such a multi-tier architecture, application servers implementing most process logic connect to a backend (central) DBMS, and the latter often becomes the bottleneck of performance. We can improve the backend DBMS by storing frequently used results in the attached cache. This technique can be regarded as “materialized view” of DBMSs. Such kind of application of cache will not contribute to the reduction of network traffic. Thus the objectives is different.

2.3 Multi-Level Store for Persistent Objects

Another relevant work is *multi-level store* of persistent objects. M. Stonebraker [12] proposed the extension of disk-resident database to include multiple storage levels, where time critical objects reside in main memory, other objects

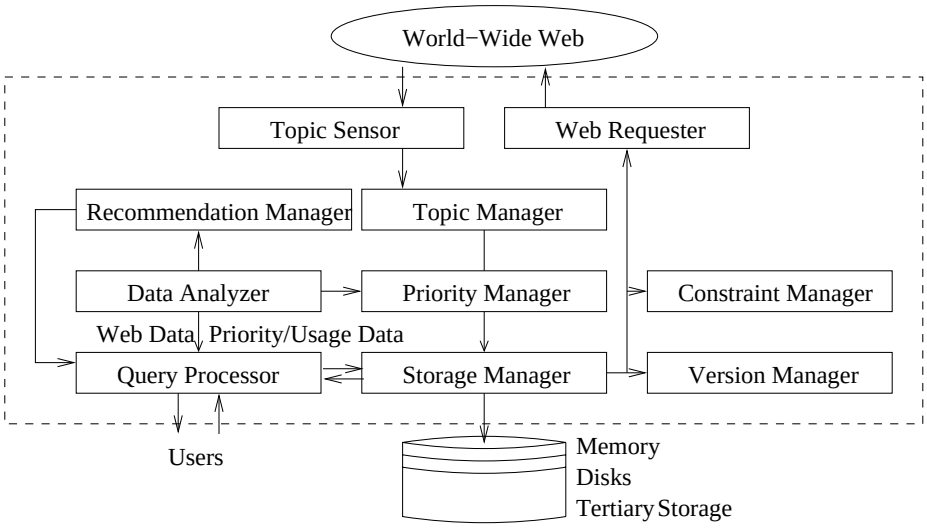

Figure 1: Architecture of a CBFWW

are disk resident, and the remainder occupy tertiary memory. It is possible that more than three levels will be present and that some of these levels will be on remote hardware. Distribution of objects in a storage hierarchy is based on semantic criteria. For example,

main memory representation: EMP where age $>= 30$ and age < 60
disk representation: age < 30
archive representation age $>= 60$

The distribution criteria can be changed dynamically by either an application or a database administrator. A specific application can temporarily redistribute instances by temporary redistribution criteria prior to execution. A database administrator can permanently change the distribution by defining permanent distribution criteria. A special program called *vacuum cleaner* is dedicated to enforcement of the distribution criteria as well as management of buffers for data from lower levels.

In our system, definitions on semantic criteria are not required. By the priority computed from the usage data, the location of objects in the storage hierarchy is determined dynamically in a self-organizing manner, although it is possible to use manual definition together by various reasons (security, for example).

3 Overview of a CBFWW System

A CBFWW can be regarded as a combination of Cache (self-organization property), Database (non-transparent usage), Search Engine (recommendation) and Data Warehouse (data analysis and mining). Figure 1 shows an overview of CBFWW architecture. Functions of components are as follows.

(1) Query Processor

A query given by a user is modified by the contents of Topic Manager. It will be transmitted to Storage Manager to find out web pages related to the query, which are stored in the systems. Corresponding page contents with priority information are obtained and transmitted to Data Analyzer. The analyzed data with the results from the storage manager are returned to Query Processor. Partial results obtained from CBFWW are given to the user. If not satisfied, the query is modified further by the result and transmitted to Web Requester to get additional contents from web.

If a user only asks analysis data of stored contents, the query is given to Storage Manager and then the results are analyzed by Data Analyzer. Other kind of query like popularity-aware queries (see section 4.3) is processed similarly.

(2) Topic Manager

By analyzing contents with priorities we can get words and phrases with weights showing the importance. Relationships between topics can also be computed using coexistence relationship. Importance of topics is determined by usage data as well as data from Topic Sensor.

(3) Topic Sensor

We have analyzed data obtained from a provider Kyoto-inet (`http://web.kyoto-inet.or.jp`). There are popular topics which have concentration of usage for rather short period. Such queries are influenced by topics given by news sites. Topic Sensor searches typical news sites to find out important topics. These topics can be used to predict future frequent queries. They can be used for modifying weights of topics managed by Topic Manager.

(4) Priority Manager

In conventional cache using a LRU strategy, a newly accessed web page have the top priority. If it is not used the priority will be decreased. In our system, we determine the priority of a page when it is retrieved, since 60% of pages will not be used again. Basic idea is to use the similarity of web pages. If a new page has many words/phrases in common with some pages that have known priority, then the same priority will be assigned to the new page. Details of this method will be discussed in Section 5.3.

Such priority will be modified by hot topics obtained by Topic Sensor. If a web page has hot topic words/phrases, the priority will be increased. Furthermore, there are interaction problems of mutually related pages to be discussed in Section 4.2

(5) Recommendation Manager

High quality contents and useful navigation paths can be obtained from usage and content mining, and used for recommendation. *Views of relevant contents* are maintained for each user so that recommendation is possible. Past usage of navigation paths is analyzed by Data Analyzer to be used for making good navigation plans. Often experienced web users, or experts can find high quality and relevant information with less efforts than less experienced ones do. Navigation that takes advantages of experiences of others is also known as "Social Navigation" [6]. CBFWW can provide such functions.

(6) Version Manager

If there is extra capacity, previous contents of web pages can be stored. A user can know the data in the past. Version Manager takes care of versions of contents. Some of important functions are discussed in the following sections.

(7) Constraint Manager

Constraints are a set of conditions the cache has to satisfy. Traditionally, capacity constraints are the most important: once this kind of constraints are invalidated, some objects have to be evicted so as to make space for worthy data. In this paper, as we assume that capacity is large enough to hold all data we like to bring in. Instead of capacity constraints, we have to take into account the following constraints.

1. **Admission Constraints** are criteria for what kind of objects are allowed to enter each hierarchy level. Typical admission constraints include, for example, the limit of object size, the limit of update frequency, and limit of copyrighted resources.

2. **Consistency Constraints** include criteria for freshness of objects in cache. Resources gathered in the system may not necessarily be consistent with the original data on servers due to updates, thus they should have to be kept up-to-date.

There are strong consistency and weak consistency. Strong consistency requires to check on each modification on either the copy or the origin. If strong consistency is imposed, the copy should synchronize with the original contents. Weak consistency can allow past data, since we have to consider usage frequency as well as average period of updates, to determine polling cycle for each object.

4 Use of Priorities for Advanced Queries and Storage Management

In this section, we will discuss how priorities are used for advanced queries and improve efficiency of query processing.

4.1 CBFWW Objects

In conventional cache, priority information is used for self-organizing contents management. Priorities are still important in CBFWW since frequently retrieved data should be available in a short time. In this section, we will describe how to manage priorities for composite data appearing in the cache.

Web data in a CBFWW can be defined as a collection of objects with priority orders $\preceq$ and a set of constraints. We can denote a set of web data handled by CBFWWs as

$$< Objects, Constraints, \preceq >$$

Each object has a unique identifier and is associated with usage information and other meta data that are automatically collected and maintained by the system. Through usage information, query and navigation can be made popularity-aware, such as retrieving most recently used objects about "data stream". Objects form a hierarchy so that data management can be done on different granularity levels. *Priorities* are defined on each level of objects in terms of meta data maintained by the system.

A Hierarchy of CBFWW Objects

As will be described later, the object hierarchy for web data is formed adaptively and consists of raw web objects, physical page objects, logical page objects as well as topics.

1. *Raw Web Objects*. Single files from web sites. The smallest units of data that our system can deal with. For example, single html files, embedded image or audio files, etc. The *Storage Manager* shown in Figure 1 deals with the raw web objects and their migration among storage devices in a storage hierarchy. It must use structural information from Physical Page Manager (to be described) and manages usage and priority information about raw web objects.

2. *Physical Page Objects (Composite Objects)*. A set of raw web objects composes a complete visual unit in a web browser. A physical page (object) often consists of a container html object and a few component objects, e.g., embedded images. The *Physical Page Manager* is responsible for management of structure/integrity and usage status for physical page objects.

3. *Logical Page Objects*. A set of one or more physical page objects forms a complete logical unit based on frequently traversed paths. The *Logical Page Manager* is responsible for management of path structure and usage status for logical page objects, supporting guided navigation when d a reference is detected towards the start point (a physical page) of a logical page path.

4. *Semantic Region Objects*. Semantically similar logical or physical page objects are clustered. For ex-

ample, a newly arrived object will be assigned it priority by the priorities of objects already in the system, which are similar to the new object. The *Semantic Region Manager* is responsible for management of construction and usage status of semantic region objects, supporting popularity-aware query to the web contents.

Hierarchy of Indices

Existence of indices will help to reduce the access time. There are the following types of indices:

- Index for raw web objects
- Index for physical page objects
- Index for logical page objects
- Index for semantic region objects
- Index for composition

Index for raw web objects (textual objects only) is generated by the words/phrases appeared in the web objects. Other indices are generated by the component indices. Index for composition stores the structural information, for example, a semantic region contain a set of logical pages. As the storage required for these indices is very big, we have to prepare an index for indices to form a index hierarchy. As indices stores in the main memory can be processed in a short time, how to determine priorities of indices is one difficult problem.

Levels of Details

The index hierarchy discussed above correspond to levels of details. If two objects A and B have identical priority, usually it is assumed that A and B are store at the same storage device in the storage hierarchy. If the size of B is very big, we may not be able to store at the same storage device. We can generate B', which only contains word/phrase information of B. Since B' is small, it can be stored at the same level as A, although B should be stored as well. B' can be regarded as an index for B.For pictures, we may be able to use pictures of low resolution.

4.2 Priority Management

CBFWW distinguishes itself from other data systems by its caching features, particularly the features of adaptation to access patterns and the priority management. Data in a CBFWW differs from data in other data systems in that history of past usage associated with each objects is maintained by the system. The associated history information reveals the access patterns and can be used to identify temporal locality of reference. Table 2 gives important attributes to quantify history of usage.

Attribute		Type	Description
frequency	f_i	int	frequency of references
firstref	t_i	time	time of first reference
lastkref	t_i^k	time	time of last k'th reference
lastkmod	u_i^k	time	time of last k'th modification
shared	r	int	number of its containers

Table 2: Attributes Representing History of Past Usage

Object Attributes for Priority Definition

Frequency f_i measures how often object i was referenced during a fixed period of time. The more often an object is accessed, the higher the probability of reuse is. Thus, objects with higher reference frequency will be given higher priority. Note, the LFU (least frequently used) cache replacement policy is just based on this attribute. Frequency of reference for an object can be computed by a few methods.

- *Sliding Window*: Computing frequency within an movable interval of fixed length (window), for example, the last week is a sliding window of size 7 days and window moves on a daily basis. To realize a sliding window, one has to keep track of detailed usage information for all data about the current window.
- $\lambda-$*Aging*: This method removes the overhead for keeping usage information. $f_{i,j} = \lambda \cdot f^* + (1-\lambda) \cdot f_{i,j-1}$, where $f_{i,j-1}$ is the (average) frequency of reference for object i at time $j-1$, f^* is the frequency since last computation. Then, $f_{i,j}$, current frequency of reference at time j is a weighted sum of the two frequency ($0 \leq \lambda \leq 1$).

Firstref t_i records the time when object i was accessed for the first time. As capacity is not considered, an object may persistently exist in the system. Modifications do not change the t_i of an existing object i.

Lastkref t_i^k is the time of the last k^{th} reference to object i. In case i has not been accessed as many as k times, $t_i^k = -\infty$. If $k = 1$, then t_i^k becomes the time since last reference of i, as used in the LRU strategy. Attributes associated with different kinds of objects are used to determine the priority of the corresponding objects.

Using Structural Information in Priority Definition

The priority of a physical page (a logical page) is determined by the maximum priority of logical pages (semantic regions) containing the physical page (the logical page, respectively).

In Figure 2, for example, physical pages D_2 and D_3 is sharing the raw web object E_5. Support D_2 and D_3 are accessed 12 and 7 times respectively in the past week. E_5 will be accessed 20 times due to the container pages' accesses. However, this may not necessarily mean E_5 is popular than D_2 or D_3. Since E_5 alone cannot be used alone, the reasonable priority of E_5 should be based on a maximal reference frequency between D_2 and D_3, which is 12 in this

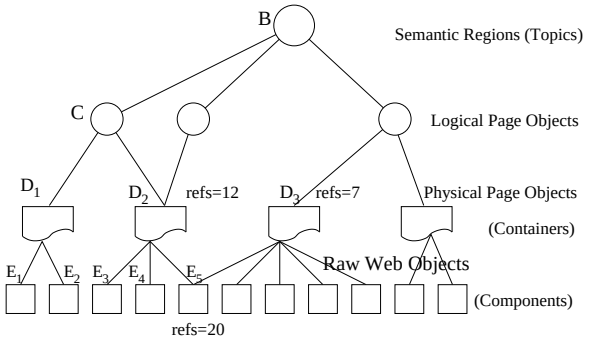

Figure 2: Object Hierarchy for (Hypertext) Web Data

example. Similarly, when measuring priority of a physical page or logical page, it is also reasonable to consider such internal structures.

4.3 Popularity-Aware Query

Direct access using an object identifier has been a unique means to use data in a cache. In this way, as aforementioned, only a very small fraction of data could luckily get accessed, whereas the majority of data will hardly be reused. It is beneficial to help users learn from other users' experience. An attractive feature of our system is that we allow users to issue queries associated with usage-based constraints, to investigate information of different popularity. Popularity-aware query is one of features that help users find information with reference to their popularity.

Query capability can be used either to facilitate cache management or to assist an end user or application to locate data associated with history/usage information. In the following, we assume an OQL-like language with several extensions to handling usage information. We assume LRU (least recently used), MRU (most recently used), LFU (least frequently used) and MFU (most frequently used) as new modifiers for filtering querying results based on their usage information. Each of them can be followed by a number that indicate how many LRU (MRU, LFU, MFU) objects to be returned. These modifiers are used the same way as DISTINCT keyword in SQL syntax.

One example of such queries is to find "most frequently used documents about "data warehouse"

```
SELECT MRU p.oid, p.title
FROM  Physical_Page p
WHERE p.title MENTION ''data ware-
house''
```

In this example, the system will first find physical objects relevant to "data warehouse", then among which choose the most frequently used one, A. As another example, to find "top 10 most frequently used logical pages that contain physical pages with sizes larger than 200(KB)", we can use the following statement.

```
SELECT MFU 10 l.oid, l.path,
FROM  Logical_Page l
WHERE EXISTS
( SELECT *
  FROM Physical_Page p
  WHERE p.oid IN l.physicals
        AND p.size > 200,000);
```

We can also query for frequent paths using logical page objects.

```
SELECT MFU, l.path
FROM  Logical_Page l
WHERE end_at(l.oid) IN
( SELECT p.oid
  FROM Physical_Page p
  WHERE p.url="http://www-
db.cs.wisc.edu/cidr/");
```

This statement tells the system to find the "most frequently used" logical pages that end at CIDR 2003 home page with a URL `http://www-db.cs.wisc.edu/cidr/`. By this way, we can find out the most popular way that users used for reaching CIDR 2003 home page.

For efficient processing of queries, proper assignment of objects to storage hierarchy is required. In general, an object with priority should be assigned to fast storage. There are the following problems. (1) Although there are many levels of priorities, the levels defined by storage hierarchy are limited. (2) There is a composite object, whose priority is related to its component objects (see Figure 2 and to be explained later). (3) There are large documents with high priority, where main memory can not store the whole documents. (4) Priority of an object will be dynamically modified.

For (1), we have to increase the levels of storage hierarchy by duplicating data in disks. Use of indexes can also improve the speed of access. Problem (2) will be discussed in the next section. For (3), we should introduce levels of details for large documents, where summary or abstract can be stored at fast storage level to provide a fast preview even the original document is currently not available. For problem (4), we should deal with the dynamic migration of objects among levels of storage hierarchy.

4.4 Self-Organizing Storage Management

As the specific characteristics of web data is the existence of hot spot data, which are usage dependent and time dependent. We have analyzed usage data obtained from a public provider Kyoto-inet. Hot spot data is very much influenced by the hot topics in news papers /TV or local events. The lifetime is very short. For example, for local events, there will be almost no access of the corresponding web pages after the event even though the event was very popular.

The major problem to be discussed is how to map data with priority to locations in storage hierarchy as shown in Figure 3. As even in the same storage in the storage hierarchy, the access speed can be different (existence of indices,

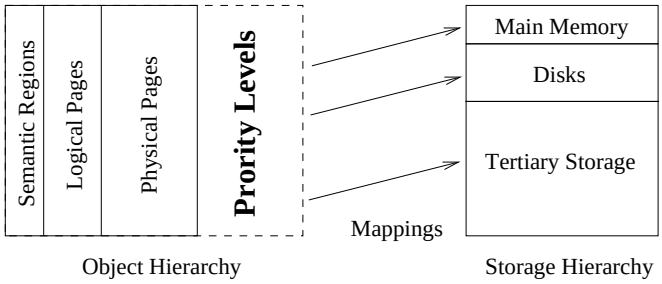

Figure 3: Mapping Object Hierarchy into Storage Hierarchy Adaptively

copies, etc.), we can realize levels corresponding to the priority levels. Some important topic are summarized here.

Locality of reference: Related objects are stored in adjacent areas of storage (disks, tapes) so that they can be retrieved together efficiently. Examples of use of this techniques are as follows.

- Several objects are required to be retrieved together due to the composition structure discussed in Section 5.1.
- Even old data have priority difference. For example, web data once in hot spot may be retrieved together for analysis purpose. Such data are clustered in the tertiary storage.

Data Migration: By the change of priority, the location of data in the storage hierarchy should be moved dynamically. To cope with recovery problem, copy control is required as follows.

- Data in main memory have exact copies in the disk.
- Data in the disk have back-up copies in the tertiary storage, which may not be exact copies due to the periodically back-up process.

By these reasons, data in the highest priority have copies in disks and (possibly old) copies in the tertiary storage. Similar solutions will occur for data in disks. For down-grading of priority, we just need to make data in the main memory invalid. For up-grading, we have to make copies. It is also necessary to maintain indices. To get up-to-date priorities and location data is not easy.

Besides self-organizing functions we also need facilities like storage schema definition language.

5 Object Hierarchy Model for Web Data

As we allow query and guided navigation on the contents of data in cache, we should investigate how web data is modeled and organized in a CBFWW. Unlike conventional cache, we have to handle interconnected objects. In addition, as a CBFWW is still a cache that always keep popular data on the top of storage hierarchy for fast access, the model should easily adapt to locality of reference. Assuming a hierarchical model is important because it is easy to identify spatial locality of reference in an object hierarchy.

We will develop a data hierarchy model for web data, in particular, the hypertext data. We believe data of other form, such as XML could also be integrated into this object hierarchy scheme.

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".

We will develop a semantic model by taking into account both structural and semantic information of hypertext data. Particularly, we define the concept of semantic regions 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.

5.1 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 [4].

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 (Figure 4). 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

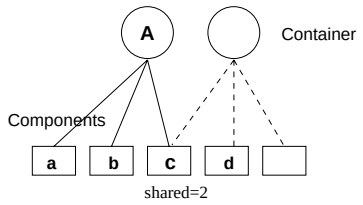

Figure 4: Document Composed by Hypermedia Components

document is called as span-to-node link. 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.

Documents can be evaluated in terms of size, recency and frequency of reference. To measure the relevance of a document to some interested "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.

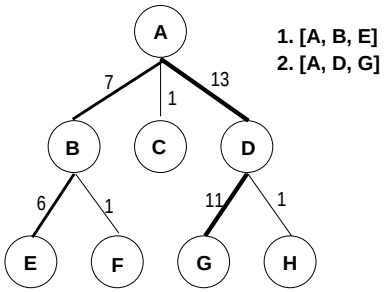

Figure 5: Logical Document Based on Repeated Traversing Paths

5.2 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.

Figure 5 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, and 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 Figure 6, suppose we 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

$$contnentof(l_i) =$$

$$< text(a_1) + text(a_2) + title(d_3), body(d_3) >$$

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 Shinkansen superexpress", the the logical document will have a logical title "Travel to Kyoto, List of bus stations, Kyoto station, Access to the Shinkansen superexpress". Note that logical

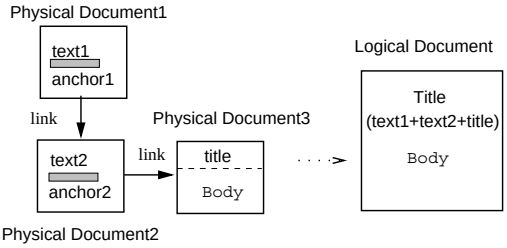

Figure 6: Link Navigations for Specific Information

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.

5.3 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).

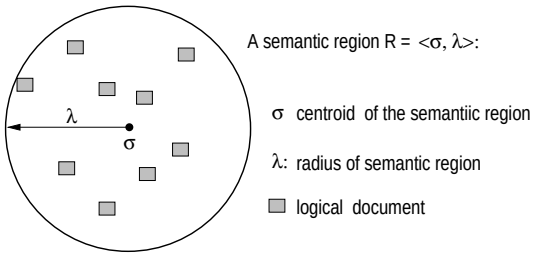

Figure 7: Semantic Region Based on Adaptive Clustering of Logical Documents

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 [14, 2, 8]. 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*[8] 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 paper, 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 Shinkansen 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.

Based on the object hierarchy model for web data, we can now outline how to manage in a content-sensitive way. Figure 8 show the situation where newly retrieved objects are assigned higher priority due to the high priorities of the associated logical pages and semantic regions. As objects with higher priority can stay in main memory, they can be accessed more quickly.

On the contrary, if an object belongs to a logical page and semantic region with lower priority, even it is currently active in use, the priority may be low and may soon migrate to lower level of storage hierarchy to make space for higher priority objects. By this way, Data in semantic regions that have been assigned higher priority can be used more efficiently than those belong to less important semantic regions. This make data and storage management in a CBFWW content-sensitive.

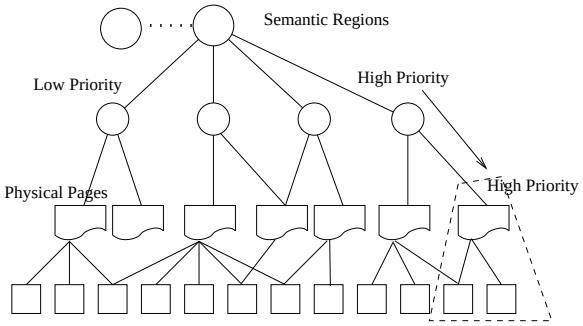

Figure 8: Priorities With Respect To Object Hierarchy for Web Data

6 Concluding Remarks

With the significant improvement of storage technology and with the wide use of caches of very large capacity such as those in CDNs, a large amount of web data has been collected at location near clients. To make full use of the history-rich information for value-added services, in this paper, we proposed a new architecture that enable popularity-aware queries and self-organizing storage management. Our long-term goal is to build a general purpose system that incorporates data management functions as in database and online decision support capability in data stream model in cooperation with dynamic hot spot data. We have been developing a prototype to be used by a provider Kyoto-inet.

Acknowledgment

This work is supported in part by JSPS (Japan Society for the Promotion of Science) and CREST of JST (Japan Science and Technology Corporation). The authors would like to thank Dr. Mukesh Mohania (IBM India Lab, New Delhi, India) for many interesting discussions.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2002)*, pages 1–12, Madison, Wisconsin, June 2002. ACM.

[2] P. S. Bradley, U. M. Fayyad, and C. 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.

[3] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997.

[4] F. G. Halasz and M. D. Schwartz. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.

[5] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. G.Lindsay, and J. F. Naughton. Middle-tier database caching for e-busines. In *Proceedings ACM SIGMOD Conference on Management of Data (SIGMOD 2002)*, pages 600–611, Madison, Wisconsin, May 2002.

[6] A. J. Munro, K. Hook, and D. Benyon, editors. *Social Navigation of Information Space*. Springer Verlag, November 1999.

[7] J. Nielsen. Nielsen's Law of Internet Bandwidth. The Alertbox, April 1998. `http://www.useit.com/alertbox/980405.html`.

[8] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *International Conference on Data Engineering (ICDE 2002)*, 2002.

[9] D. A. Patterson and K. K. Keeton. Hardware technology trends and database opportunities. In *Proceedings ACM SIGMOD Conference on Management of Data (SIGMOD 1998)*, Seattle, Washington, June 1998. Keynote Address.

[10] L. Rizzo and L. Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, 2000.

[11] L. G. Roberts. Beyond Moore's Law: Internet growth trends. *IEEE Computer–Internet Watch*, 33(1):117–119, Jan. 2000.

[12] M. Stonebraker. Managing persistent objects in a multi-level store. In *Proceedings ACM SIGMOD Conference on Management of Data (SIGMOD 1991)*, pages 2–11, Denver, CO, May 1991.

[13] T. Team. Mid-tier caching: The timesten approac. In *Proceedings ACM SIGMOD Conference on Management of Data (SIGMOD 2002)*, pages 588–593, Madison, Wisconsin, May 2002.

[14] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings International SIGMOD Conference on Management of Data (SIGMOD 1996)*, pages 103–114, 1996.