Structural Caching XML data for Wireless Accesses

Shiu Hin Wang
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
852-97235397
hwshiu@mail.hongkong.com

Vincent Ng
Department of Computing
The Hong Kong Polytechnic University
Hong Kong
852-27667242
cstyng@comp.polyu.edu.hk

Abstract

Recent web cache replacement policies incorporate information such as document size, frequency, and age in the decision process. In this paper, we propose a new caching algorithm, StructCache, for wireless accesses of XML data. The algorithm is an enhancement of the Greedy-Dual-Size (GDS) policy and the Greedy-Dual-Frequency-Size (GDFS) policy. It would consider document sizes, access frequency and exploits the aging mechanism to deal with cache pollution. In addition, the structural information of XML is utilized to achieve better hit ratios. Experimental results show that the StructCache algorithm outperforms GDS and GDFS algorithms for queries which are sub-tree(s) in XML documents of precedent queries and queries of the same axis and node tests in XML documents with precedent queries.

1. Introduction

By virtue of the increasing processing power of embedded computers, wireless computing and Mobile Commerce (mCommerce) is the wave of the future [7]. Caching and prefetching of XML data in heterogeneous networks, especially for the mobile environment, reduce traffics and improve the performance of dissemination of XML data, which in turns improve the usability of the Internet as a large and distributed information system.

Very often, user access patterns are helpful for the customization for specific type of users. The relative importance of long-term popularity and short-term temporal correlation of references for web cache replacement policies has not studied thoroughly. This is partially due to the lack of accurate characterization of temporal locality that enables the identification of the relative strengths of these two sources of temporal locality in a reference stream [15].

Moreover, better cache policies are equivalent to several-fold increase in cache size. Efficient cache and prefetching algorithms reduce the needs of cache sizes to match the growth rate of web content. The gains from efficient cache and prefetching algorithms are compounded through a hierarchy of caches [15].

Existing web caching algorithms capture the characteristics and differences of paging in file systems. However, they do not consider the nature and properties of the objects themselves. In this paper, we try to propose a caching technique to improve the performance of query responses. It is our aim to improve the performance of XML queries against large XML files, which in turn may improve the usability of wireless applications.

In this paper, our main focus is the benefits brought from our proposed replacement algorithm to cache XML documents and the comparison of performance with other algorithms. The paper is composed of six sections. In section 2, we first review background study and previous related work. Section 3 consists of the structure and details our proposed XMLCache framework. Section 4 gives the details of our caching algorithm, StructCache, for caching of XML objects. The procedures and results of the experiment are presented in section 5. Section 6 summarizes our work and section 7 contains the references.

2. Background and Related Work

Existing web caching algorithms mainly consider individual documents as the individual objects to be cached. The larger the document, the greater the overhead when cache misses. This may pose a problem especially under a bandwidth and memory constrained environments such as wireless environments.

Traditional object caching algorithms that have not considered the syntactic and semantics characteristics of XML documents may not handle the HTML, XML contents in an efficient manner. Our suggested caching algorithms tries to exploit the syntactic structure of XML documents and the XML based quires to improve the caching performance in a high latency and low bandwidth environments.

We develop a caching framework that is used for caching of both XML and non-XML documents. The caching technique is done on client-side, which is supposed to be embedded in wireless computing devices with limited bandwidth communication connections. We will study the performance of our proposed caching algorithm that tries to exploit the schemas of XML and XML queries. It is also expected that the algorithm will be more effective, especially in situations in which the object size to cache size ratio is high, high network latency and low transmission environment.

2.1. Cost Metrics

Cost metrics [1,5,6] are used as objective measurements of the effectiveness of the caching algorithms. Three cost metrics mostly employed are:

1) Bit Model: The cost of a cache miss equals to the size of the missing item. This measure provides an objective measure for the effectiveness of our proposed algorithms.

2) Cost Model: The cost of a cache miss is unity. This is used for evaluating the use of the heuristics of XML queries in improving the effectiveness of caching.

3) Time Model: The cost of a miss equals to the average time to load such an individual item. Here, it means the time to retrieve the whole object (time of retrieval due to page fault) or the user perceived response time (network Delay). For some wireless application, a user who has part of the results and progressively getting the remainings may be more crucial than obtaining the complete results at a minimum time even though the total time of retrieval is longer.

2.2. Related Work

Caching algorithms targeting for web have become more prevalent. GreedyDual [15] web caching algorithm is one of the typical examples. It is a generalization of GreedyDual-Size algorithm [5] and a development of the family of algorithms derived such as GreedyDual Frequency-Size. Trace driven simulation illustrates that it has superior performance when compared to other web cache replacement policies proposed in the literature[15].

There are many factors affecting the performance of a given cache replacement policy. GreedyDual caching algorithm exploits the size, miss penalty, temporary locality and long-term access frequency and captures both popularity and temporal correlation:

1) Size – Web objects are of various size and caching smaller objects usually results in higher hit ratios, especially given the preference for small objects [16].

2) Miss Penalty – The miss penalty varies significantly. Assigning higher preference to objects with a high retrieval latency can achieve high latency saving [14]

3) Temporary locality – Web access patterns exhibit the temporal locality [2]. Similar to the LRU replacement policies, GreedyDual also assign higher preference to recently accessed objects.

4) Long-term access frequency – The bursty behavior of the popularity of web objects were found over short times scales while it is more smooth over long times scales [3].

Apart from the researches in replacement algorithms targeting for Web, there are hierarchical and cooperative caching architectures such as the Harvest project [8], access driven cache [9], adaptive Web caching [10]. For distributed caching architecture, there are examples such as summary cache [11] and Internet cache protocol (ICP) [13]. With hierarchical caching, caches are placed at multiple levels of the network. A hierarchical architecture is more bandwidth efficient, particularly when some cooperating cache servers do not have high-speed connectivity. With distributed caching, most of the traffic flows through low network levels, which allows better load sharing and are more fault tolerable. However, a large-scale deployment of distributed caching may encounter the problems of high connection times, higher bandwidth usage and administrative issues [12].

Because of the relatively small bandwidth of mobile environment when compared with that of connected networks, the gains arising from efficient caching and prefetching in mobile environments are even apparent. However, conventional replacement algorithms still have room for improvements under the Internet and mobile environment when data objects are having syntactic and semantic implications.

3. The XMLCache Framewok

Our proposed framework utilizes the conventional caching algorithms for non-XML documents whilst handles XML data objects differently. The determination of which caching strategy is employed depends on the type of documents requests from clients. A different caching algorithm (StrcutCache) will exploit the coherency of similar requests made by clients with the idea of the structure of XML documents will have impacts on successive retrieval.

The framework shown in Figure 1 acts as a proxy between the mobile clients and remote data source. It mainly divides retrieval objects into XML and non-XML objects. For XML data queries, it is handled by our proposed caching algorithm called StructCache in the caching engine. For non-XML data queries, it is handled by existing web caching algorithms. Depending on whether the XML documents fetched from remote servers have the corresponding DTD (Document Type Definition) or not, the DTD may need to be extracted by the DTD Extractor.

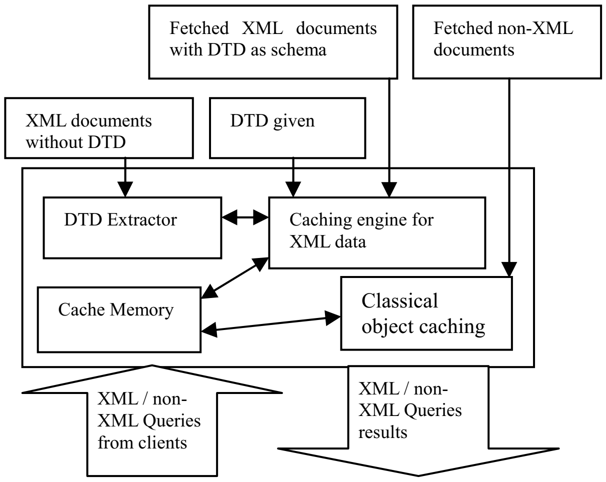
Fetched XML documents with DTD as schema
Fetched non-XML documents
XML documents without DTD
DTD given
DTD Extractor
Caching engine for XML data
Cache Memory
Classical object caching
XML / non-XML Queries from clients
XML / non-XML Queries results

Figure 1. Internal architecture of StructCache framework

Every XML query sent from clients is evaluated, which triggers the retrieval of XML documents when the required fragments do not exist in the cache memory. The client returns the query result directly when the required objects are located in cache. Each data object of XML documents in cache memory is associated with a timestamp and the corresponding DTD. DTDs of the retrieved XML documents do not exist will trigger the generation of initial multiplication factors, which will affect the score updating process in later phase. Depending on the frequency of accesses and sizes of objects, a score for each of the data node of a given DTD will be updated in runtime. More details will be covered in Section 4.3.

In our framework, there are a number of assumptions:

a) Client computation power, power consumption and the complexity of the algorithm are not the limiting factors.

b) Since the cache is located on the client wireless devices (e.g. PDA), the size of the cache as well as the bandwidth of the wireless networks are the limiting factors.

c) Because of the high latency, and relatively low transmission rate of the communication channels, the overhead of retrieving objects from heterogeneous network is higher than that from the cache. We assume that the throughput and the latency are averaged constants throughout our model. We choose a typical transmission rate reported by mobile network services providers rather than that derived theoretically from a selected modulation methods, and carriers.

d) We assume caching can be performed either in the mobile clients or proxies. Details of switching of underlying cellular networks are intentionally abstracted to generalize our proposed algorithm for different kinds of carriers or application-specific uses.

3.1. Queries from clients

In general, requests from mobile computing devices expressed in URL-like query can be decomposed into two parts. The first part is the URL that locates the resource to be retrieved whilst the second part is an XPath expression. Figure 2 illustrates an example of a client request. In this example, the first part is the 'http://www.polyu.edu.hk?/article/author/name' and the second part is '[firstname='Joe']'.

The syntax of the client request:
[URL]?[XQL]
Key:
[URL] : Universal Resource Locator that locates the resources(XML / non-XML files)
[XQL] : XQL of the target XML document
Example:
http://www.polyu.edu.hk?/article/author/name[firstname='Joe']

Figure 2. An example request from client application

Our cache engine acts as a proxy between the client application and remote servers. Items fetched on behalf of client applications include XML and non-XML files. The fetched XML documents can also be classified into two types. The first type is those XML documents have the corresponding DTD given in advance` whilst the second type has no knowledge about the document's DTD. In the first case, our replacement algorithm can be directly applied. For the second case, since the corresponding DTD of a XML document not known in advance, a DTD extractor is used for the estimation of the structure of the DTD.

The XMLCache framework has 4 major modules:

a) DTD Extractor – This module is responsible for the extraction of DTD from a given XML document no DTD given in advance. It is employed when clients requests query XML documents that have no predefined DTD.

b) The StructCache Engine – Our proposed engine of caching XML data under mobile.

c) Cache Memory – The module has two parts. The first part is a set of mappings that maps the document identifier (URI) to a given timestamp (Document's Last Modification Time). The second part is the repository where individual cached objects are placed. For XML documents, it should be the fragments of XML documents whilst it will be complete binary image for graphics files.

d) Classical Cache Engine – This component encompasses conventional online replacement algorithms such as GDFS, which treats the whole file as an individual object to be cached. This is mainly used for caching of non-XML documents.

4. StructCache

StructCache is an algorithm used in the caching engine for XML data with DTD. The algorithm can be divided into two phases. The first phase is to determine the weighting factors, called multiplication factors for a group of XML documents with the same schemas which are used in runtime phase. During operations, the score of each node, which is depended on the multiplication factors calculated in the first phase, access frequencies, size of nodes as well as the XPath of the XML queries derived from subsequent client requests.

Comparing with other classical object caching algorithms, StructCache adopts an adaptive way rather than solely relying on frequency, size of objects in cache. It is adaptive in the sense that the initial multiplication factors depend on the structure of the associated DTD, and the object replacements process is affected by the multiplication factors, XPath, the heuristics of access of nodes or nodes sets as well as the size of a given node or nodes sets relative to the whole XML document. Although we do not quantify the relationship between the factors considered and their effects in this paper, it is observed that the factors concerned are correlated with long-term access patterns. It is because, very often, the design of schema mostly reflects the relative importance of fragments in XML documents and the temporal and spatial locality of accesses.

Normally, systems can either fetch a block in response to a cache miss (on-demand fetch), or it can fetch a block before it is referenced in anticipation of a miss (prefetch)[16]. Fine-grained objects will save more redundant space but sacrifice the computation power. Our goal is to find an online policy for on-demand fetching and caching of XML documents without knowing the sequences of references in advance. Instead of caching the whole XML document, the memory objects to be cached are fragments of XML documents derived from the results of XML queries in the request stream, which may be of various sizes. For a limited amount of cache in wireless computing devices, we try to exploit the heuristic as well as the semi-structured characteristics of XML documents. Our problem in fact is a general caching problem [12], when the pages have varying sizes and costs. This problem arises, among other places, in cache design for networked file systems or the world-wide web [12]. In web caching, popular online caching algorithms such as Least Recently Used (LRU) has heuristics justification that real-life sequences often exhibit the property that "the past predicts the future". They normally treats the whole file as an individual object [1]; by contrast, we try to apply a more fine-grained approach such that individual objects to be cached are XML fragments from the query results, with a view to minimize page faults, especially for wireless communication channels and documents with large size. The heuristics of our approach are based on the XML data queries generated by the client application(s) or requested from users.

StructCache is our caching algorithm dedicatedly for XML data with DTD. It is to work within the caching engine for XML data in our framework. The algorithm has two phases. The first phase is the determination of factors and initialization of variables from a newly fetched DTD. It is triggered by the cache miss and fetching of XML documents with undetermined DTD from remote server(s). The second phase is the runtime phase that fetches objects, invalidates cache, performs updating of variables and triggering the initialization phase for any fetched and undetermined DTD.

```
<!ELEMENT article(category, title, publisher, author*)>
<!ENTITY % Address "(#PCDATA)">
<!ELEMENT title(#PCDATA)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT publisher(publishername,address)>
<!ELEMENT publishername(#PCDATA)>
<!ELEMENT address(#PCDATA)>
<!ELEMENT author(name,age,address?)>
<!ELEMENT name(firstname?,lastname?)>
<!ELEMENT age(#PCDATA)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname(#PCDATA)>
```

Figure 3. An example DTD

4.1. Initialization Phase

In this phase, multiplication factors are generated for each of the XML, which may affects the cost of updating and in turns that of replacement. The construction of the multiplication factors is determined by a DTD of XML document, which is then stored for use in later phase.

Each node of XML documents is associated with a score, which is initialized to zero and affected by the access patterns during runtime. The initial cache plan may affect the performance of caching as the cost updating process in runtime phase depends on the multiplication factors generated in this phase.

```
<article>
        <title>
        A Relational Model for Large Shared Data Banks
        </title>
        <publisher>
                <publishername>HKPolyU Publishing Co. Ltd.
                </publishername>
                <address>Honghom, Kowloon
                </address>
        </publisher>
        <author>
                <name>
                        <firstname>E.F.</firstname>
                        <lastname>Codd</lastname>
                </name>
                <age>54</age>
        </author>
</article>
```

Figure 4. An example XML document

The initial phase consists of three steps. The first step is the construction a directed graph from a DTD of that XML documents. By expanding all entities definitions within the directed graph, a tree is generated. Figure 3 shows a sample DTD and Figure 4 shows an instance of the DTD. Figure 5 shows the tree constructed from the DTD.

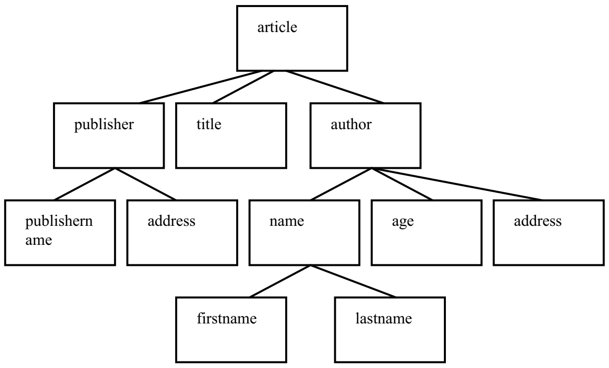

Figure 5. A directed graph constructed from the DTD

The second step is the assignment of weightings to each of the leaf nodes of the directed graph. Table 1 shows the relative weightings of properties observed from common DTDs for XML documents. The weights represent the relative strength of the relationship among nodes of a DTD from design view. The higher is the weighting, the stronger is the relationship.

The assignment of multiplier and replacement cost is based on the expected probabilities of occurrences of those nodes in instances of the DTD. Basic replacement cost depends on the occurrence notation of a node. The cost implies the relatively importance of that node derived from the DTD whilst multiplier is a factor that depends on the elements' content specifications.

We model the relationship with three kinds of factors. The first one is the common design characteristics of a DTD. An example is that a node with mandatory notation is more important than optional one. The second one is the probable inter-relationship among nodes. In this paper, we classify this kind of relationship into mutual exclusive, co-occurrence, sub-typing, and no relationship at all. The third one is the relative size of the actual instantiation of the nodes. Therefore, the model of relationship can be constructed as:

$$R \sim (T + I) / S$$

where R is the relative strength of importance, T is the common design characteristics of a DTD, I is the type of inter-relationship, and S is the relative size of the probable instantiation of a node.

For easier operation, the size of the probable instantiation of a node is replaced with the number of options of that node and the common design characteristics into either no implication, optional or mandatory, which is normalized to 1, 1/2 and 1 respectively. For the type of inter-relationship, we normalize the co-occurrence, sub-typing, no relationship and mutual exclusive relationship into numerical values 1, 1, 1/2, 1/n where n is the number of options.

Table 1. Assignment of weighting factors

Element with Occurrence Notation	Basic replacement cost(Q)
?(Optional)	1/2
*(Zero or More)	1
+(One or More)	number of options
|(OR)	1/number of options
No Notation	1

ATTLIST	Basic replacement cost(Q)
Element(s) with attribute ID	1
Element(s) with fixed attribute	1

Element's ContentSpec	Multiplier(T)
PI	1
ANY	1
Mixed	Sum of all possible weightings/number of options
Comment	1/2
Fixed	2
PCDATA	2

Basic replacement cost (Q_i) is multiplied by the corresponding multiplier(T_i) to obtain the multiplication factor that leaf node. Assignment process is then performed in a bottom-up manner. The score of a non-leaf node is the aggregated sum of its descendent nodes multiplied by the multiplier of that node. The assignment process iterates until the root node is reached. The multiplication factor for a given node W_1 is:

$$\begin{aligned} W_i &= T_i * Q_i \\ &= T_i * \sum W_{i-1} \\ &= T_i * \sum W_{i-1} * \sum W_{i-2} * \ldots\ldots * \sum W_1 \end{aligned}$$

The multiplication factor reflects the relatively importance of a given node within a DTD for a given XML document. The value of a factor derived from the DTD is a numerical representation for manipulation in the later phase. It is expected that nodes having higher values are more important as they appear more often in instances of that DTD. Figure 6 illustrates the weighted directed graph of the example DTD.

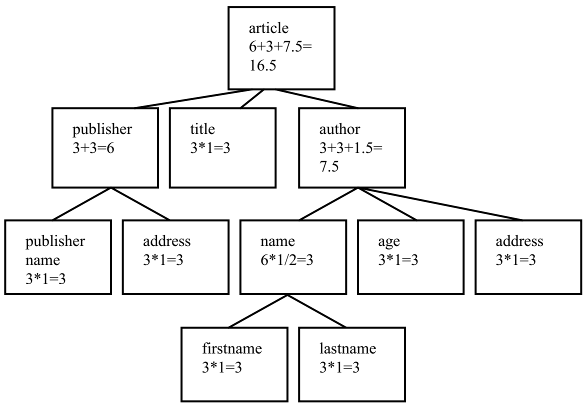

Figure 6. The weighted directed graph constructed from DTD

In step 3, the multiplication factors constructed in step 2 are derived. Apart from these factors, each node of the selected XML document has an associated score. Although both the score and multiplication factor are derived and initialized in this phase and used afterwards, the differences between them is that the latter reflects the schema characteristics of the DTDs while the former is a set of variables for manipulation in runtime phase. The associated score of each node C_i is initialized to zero for each DTD fetched from remote server.

$\forall C_i \{C_i = 0\}$

As such, a mapping of DTD and the multiplication factors, in additional to the set of initialized scores are generated in this phase.

4.2. Runtime Phase

4.2.1. Score Updating

For each of the XML query requested by the client, the timestamp of the corresponding local document is compared with that of remote server to check for data coherency. The invalidation of data triggered by the discrepancies of these two timestamps results in the retransmission of the corresponding XML document and updating of local timestamp. For a cache hit, the score of the selected node(s) C_i is/are re-calculated by the corresponding adjustment factor δ_i:

$C_i = C_i + \delta_i$

The updating of cost in each node is affected by the size of the element, access frequency and the difference between fan-out and fan-in of a node within the actual XML instantiations of the DTD. Hence, the cost, δ_i is defined as:

$$\delta_i = \alpha * \ln(W_i * S/S_i * F_i/F * E_i/E)$$

where a is a constant, W_i is the Multiplication Factor derived from initial phase, S_i is the size of node(s), S is size of the XML object fragment, F_i is the access count of that node in the cache, F is the total hit count of the XML object fragment, and E_i is a fan-out factor which is determined by the number of edges connected to children nodes U_i and that of parent nodes V_i:

$$E_j = U_i - V_i \text{ iff } U_i - V_i >= 0$$
$$E_j = 1 \text{ if } U_i - V_i < 0$$

All other unaffected nodes $\{C_i\}$ are deducted by an adjustment factor δ_j:

$C_j = C_j - \delta_j$

The adjustment factor δ_j of unaffected node is determined by:

$\delta_j = \sum\delta_i / |C_j|$

4.2.2. Object Replacement

Object replacement process encompasses three steps:

a) When the required XML document fragments are stored in the cache memory, the local copy is returned to client and no object replacement occurs.

b) Whenever cache miss or cache incoherency occurs, the requested object(s) retrieved from remote servers will be stored in local cache memory as long as the maximum available cache size is not exceeded.

c) Whenever cache miss or cache incoherency occurs and the space available in local cache memory is not enough to accommodate the requested object(s) retrieved from remote servers, object replacement process will begin and each queries will be evaluated by accumulating the score of the nodes(C_i) across the axis for the corresponding XPath. The cached query and object pair having the least evaluated total value will be evicted and the process iterates until the space available can accommodate the executed query and fetched XML fragment.

In other words, the following two criteria must be met for the occurrence of object replacement:

(i) $\forall C_k \{\sum C_i > \sum C_k\}$

(ii) $\sum Size(C_i) <= \sum Size(C_k)$

where $C_i \in$ {sets of the retrieved nodes} and $C_k \in$ {sets of the nodes of the path expression to be replaced}. Figure 7 is the summary of our proposed algorithm StructCache:

```
Find $W_i$ for all nodes of a given XML document
$C_i \leftarrow 0.0$
For each request query p do
    If p is in cache
        then
        $C_i = C_i + \delta_i$  where $\delta_i = \alpha * \ln(W_i * S/S_i * F_i/F * E_i/E)$ for all affected nodes
        $C_j = C_j - \delta_j$  where $\delta_j = \sum\delta_j / |C_i|$ for other nodes
    else fetch p
    While there is not enough free cache for p
    Evict fragment(s) with min{$\sum C_i (q)|q$} are the nodes of the axis of XQL data query
    in cache}
```

Figure 7. Summary of StructCache

GDFS employs the dynamic aging mechanism or inflation value to simulate the reference correlation of web traffic. Instead, our proposed algorithm uses the reward and punishment mechanism in updating and does not need to determine the base value during the reset step of cache hit. Utility value reflects the normalized expected cost saving if the object stays in the cache. Given the long-term reference pattern is stable, GDFS uses f(p) * c(p) / s(p) that consider the reference count, cost of fetching and size of object as well as the aging factor to approximate the utility value. By contrast, the StructCache algorithm considers the structure of XML document, in additional to the temporal locality, spatial locality, cost of fetching and size of object.

5. Experiment Results

To facilitate our evaluations of different caching strategies, we have performed a simulation to test the performance of StructCache against the GDS and GDFS algorithms. We model the clients' requests of XML by a list of predefined XML objects queries, which is executed sequentially. Different algorithms are implemented in proxy between the client and remote servers. The proxy is responsible for handling the clients' requests and returning the results to clients. In this experiment, our focus is on the performance gain by caching the queries' result of XML fragments instead of whole documents and the study the effects of the incorporation of the structure of DTD and the XPath in replacement algorithms.

During the experiment, the execution sequence of the batch of queries remains unchanged throughout the experiment. In our experiment, the predefined queries can be classified into the following four types:

1) Queries have similar XPath but different predicates.
/article/author/name[firstname='Peter'] and /article/author/name[firstname='Tom'] are example of this type of queries

2) Queries have results that are subsets of results of previous queries
/article/author and /article are example of this type of queries

3) Queries randomly select nodes and have no predicate
/article/author/name/firstname and /article/publisher/publishername are example of this type of queries

4) Queries randomly select nodes and have arbitrary predicates
/article/author/name[firstname='Tom'] and /article/publisher[publishername='ABC Publisher'] are example of this type of queries

We compare the effectiveness and relative gain of performance of StructCache with GreedyDual Size(GDS) and GreedyDual Frequency-Size (GDFS) algorithms in terms of the Bit Model and the Cost Model.

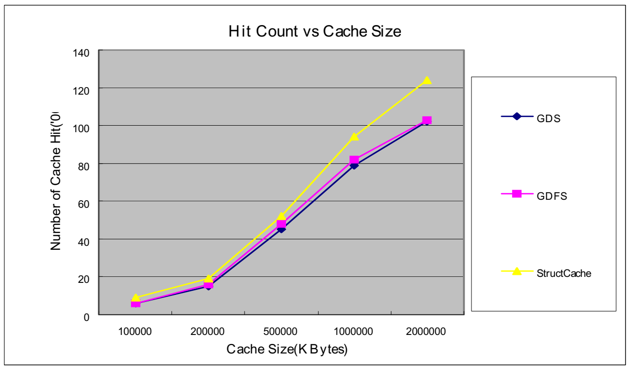

Figure 8. Hit counts versus cache Size for different evaluated algorithms

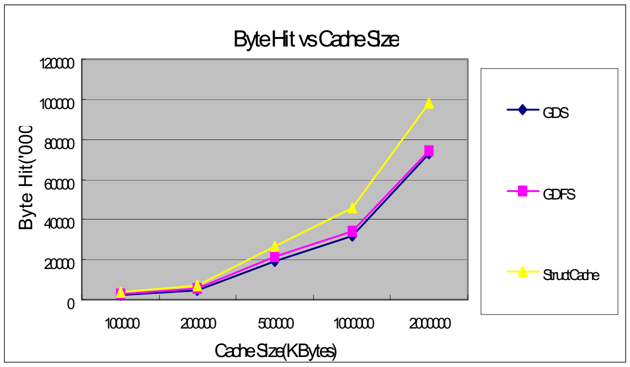

Figure 9. Byte hit versus cache Size for different evaluated algorithms

Figure 8 shows the plot of hit counts versus cache size for the three algorithms. Figure 9 shows the plot of the corresponding byte hit versus the cache size. Both results illustrate that larger cache sizes give higher hit counts and byte hits and the StructCache algorithm outperforms the GDS and GDFS algorithm in terms of the Bit Model and Cost Model for XQL queries. The improvement in hit count is up to 20% and 22% in byte hit. The gain of performance is highly related to the types of queries. The result is particularly apparent for XML documents with relatively large document size to cache size ratio. For retrieving of large size XML documents, the fetch cost is relatively large and caching of XML fragments not only reduces the size of cache objects, but also reduces the page faults.

The incorporation of syntactic features of DTD as a parameter in cost updating function and replacement algorithm gives additional information about the objects to be cached. One reason is the design of schema usually considers the relatively importance of a node and the relationship among various nodes. For the sub-typing relationship and the occurrence notation can be exploited. The stream of XQL queries are also exploited by our proposed algorithm. In StructCache algorithm, the XPath of XQL queries are used in score updating and the determination of objects replacement. We found that the following two kinds of XQL queries are well handled with our proposed algorithm:

a) Subsequent queries are specific sub-tree(s) of precedent queries

b) Subsequent queries are of the same level and path with precedent queries

In other words, it performs well when the stream of queries exhibits the spatial locality characteristics and the user access preference is 'moving from general to specific'. Results also indicate that the performance is still comparable to traditional caching algorithms even though the two above criteria cannot be met.

6. Conclusion

In this paper, we present a XMLCache caching framework for XML data under the mobile environment. It takes care of both XML and non-XML data and the replacement algorithm considers the syntactic characteristics of the XML schema in additional to the access pattern of XML queries, the long-term access frequencies and fragment size. By using the Cost Model and the Hit Model as the metrics, preliminary experiments show that our proposed algorithm outperforms the GDS and GDFS for the same configurations of cache sizes and user queries.

Acknowledgement

The work reported in this paper was partially supported by Hong Kong CERG Grant – PolyU 5094/00E.

7. References

[1] Saied Hossenini-Khayat, "Replacement algorithms for object caching", Proceedings of the ACM symposium on Applied Computing, Atlanta, GA USA, Mar 1998.

[2] R. Wooster and N. Abrams. "Proxy caching that estimates page load delays". In Proceedings of the 6th International WWW Conference, 1997.

[3] Steve D. Gribble and Eric A. Brewer. "System design issues for Internet middleware services : Deductions from a large client trace". In Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, 1997.

[4] Pei Cao, Edward W. Felten, Anna R. Karlin and Kai Li. "A study of integrated prefetching and caching strategies", Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, May 1995.

[5] Susanne Albers, Sanjeev Arora and Sanjeev Khanna. "Page Replacement for General Caching Problems", In Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms, 1999.

[6] S. Irani. "Page replacement with mult-size pages and applications to Web caching". Proceedings 29th Annual ACM Symposium on Theory of Computing, 701-710, 1997.

[7] Ed Sutherland. "Predicting M-Commerce Trends for 2002: Part II" . http://www.mcommercetimes.com/Industry/211, Jan 2002.

[8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and J. Worrel, "A hierarchical Internet Object Cache", Usenix'96, January 1996.

[9] J. Yang, W. Wang, R. Muntz, and J. Wang, "Access Driven Web Caching", UCLA Technical Report #990007.

[10] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd and V. Jacobson, "Adaptive Web Caching: towards a new caching architecture", Computer Network and ISDN Systems, November 1998.

[11] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", IEEE/ACM Transactions on Networking, Vol. 8 No.3, June 2000.

[12] Jia Wang, "A Survey of Web Caching Schemes for the Internet", Cornell Network Research Group(C/NRG), 2000.

[13] D. Wessels and K. Claffy, "Internet Cache Protocol(ICP)", version 2, RFC 2186.

[14] Anja Feldmann, Ram?n Cáceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments". AT&T Labs-Research, Florha Park, NJ, USA, 1999.

[15] Shudong Jin and Azer Bestavros, "GreedyDual Web Caching Algorithm - Exploiting the Two Sources of Temporal Locality in Web Request Streams", Boston University, 2000.

[16] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. "Characterizing Reference Locality in the WWW". Department of Computer Science, Boston University, 1996.