DOMProxy: Enabling Dynamic-Content Front-end Web Caching

Manolis Veliskakis, John Roussos, Panos Georgantas, Timos Sellis

Knowledge and Database Systems Laboratory
School of Electrical and Computer Engineering
National Technical University of Athens
{manolis, iroussos, pgeor, timos}@dblab.ece.ntua.gr

Abstract

Proxy caching is a commonly accepted methodology used to reduce Internet traffic, decrease back-end related user delays, and generally improve web performance. Nevertheless, it seems to be inadequate when it comes to caching dynamic web pages, due to their low degree of reusability and strong dependency on the back-end site infrastructure. In order to overcome these drawbacks we present DOMProxy (Dynamic Objects Manager Proxy), a new Proxy architecture that enables front-end web caching for dynamically generated web pages. Instead of caching web pages produced dynamically or their fragments, the proposed architecture provides methods for “caching” their generation process. This is achieved by caching both the applications and the back-end data content that are used to create dynamic web pages. We discuss the new challenges and issues raised by this approach, with special emphasis on the caching and manipulation of different types of web objects at a front-end caching point.

1. Introduction

Proxy caching is a commonly accepted methodology used to reduce Internet traffic, decrease back-end related user delays and generally improve web performance. Numerous web caching approaches have been proposed concerning static web pages. Given that the use of dynamically generated web pages increases continuously, there is need for web caching algorithms that deal with dynamic web pages as well.

Web caching techniques can be characterized as back-end or front-end. Back-end approaches are deployed inside the site infrastructure. They can significantly reduce content generation delays, yet they do not address network-related delays and bandwidth consumption issues. On the other hand, front-end approaches concern caching outside the site infrastructure, e.g. a proxy or a cache that resides at the edge of a Content Delivery Network (CDN). Front-end approaches are based on serving the clients’ HTTP requests before they reach the site components, dealing with both network-related and back-end delays.

Front-end web caching techniques have difficulty in handling dynamic web pages due to their strong dependency on the back-end site infrastructure. Dynamic web pages are created on request by application programs (jsp, asp etc) stored in the back-end site infrastructure. These programs usually connect to back-end databases; hence, dynamic web page creation assumes the existence of both an application program and back-end content.

Caching dynamic web pages can be performed in order to avoid the ‘creation on request’ approach. For caching to be effective in this case, a high degree of cached objects’ reusability would be needed, a prerequisite that dynamic web pages do not fulfill. The request of a dynamic web page depends on client-defined input parameters. Serving such a request using a cache could be possible only if a cached dynamic web page had been produced from the same application with the exact same input parameters, a rather rare situation. Even in that case, the creation of dynamic web pages may depend on client-related information (e.g. through the use of cookies). Thus, different calls to the same application with the same parameters may produce different output.

In summary, the reasons that front-end caching approaches have difficulty in handling dynamic web pages are (a) their strong dependency on the back-end site components, and (b) their low degree of reusability.

In order to overcome these obstacles we believe that front-end web caching approaches should be based on caching the generation process of the dynamic web pages and not the pages themselves. We present an implemented new proxy architecture, DOMProxy that follows the above web caching technique. In addition, we discuss new challenges and issues raised by this approach, with special emphasis on the caching and
manipulation of different types of web objects at a front-end caching point.

2. Related Work

Many front-end web caching approaches have focused on caching only the back-end database content. There are two main categories on database content caching. The first one includes approaches that are based on caching and manipulating query results used by application programs [1,2,8], while the other category concerns approaches that use full-fledged database caching techniques [3,4,6]. Semantic Data Caching [8] is a solution that defines ways on how a client proxy is able to cache the query results that a back-end database server produces by exploiting the query’s semantic description. An edge-of-network semantic data cache, DBProxy, is proposed in [2]; DBProxy can be considered as an improvement of Semantic Data Caching because it supports more types of transactional web queries and at the same time proposes a sophisticated algorithm for the consistency of the cached objects. Furthermore, the Proxy-Caching Framework presented in [1] can be considered as a simplified version of Semantic Data Caching: it exploits the semantics of queries expressed through HTML forms in order to achieve traditional passive query matching or active query matching.

In a different direction, in [3] and [6] database caching is done by using materialized views, replication techniques and distributed query execution. In [4] a full-fledged database server that is co-located with the application server is proposed as a solution to caching data from the back-end server. It tries to perform an on-demand caching operation by introducing a new type of database objects, called Cache Tables, which enables caching of full or partial content of the corresponding back-end database table.

Although caching database content outside a site’s infrastructure is a promising and effective technique, it cannot be considered complete. This is because it caches only one of the components used in the dynamic web pages generation process. Without the operation of the application programs, which determine the presentation and business logic of a dynamic web page, a database cache could not satisfy alone a client’s request. Approaches like [5, 9, 7] try to overcome this obstacle by giving minimum application logic to caching spots outside the site’s infrastructure (e.g. Forward or Standard proxy). More specifically, Cache Applets [5] are basically software modules written in Java, attached to the dynamic documents and sent as responses from the back-end server to clients’ requests. Through Cache Applets, web servers can control the caching behaviour of the proxies, as far as their dynamic content is concerned. One use of the Cache Applets approach is presented in [9], where the proxy is enhanced with the ability to perform query caching. This is done through a cache applet that is sent from the web server and gives query-processing capabilities to the proxy. Finally, in [7], the Dynamic Content Caching Protocol (DCCP) is proposed; by exploiting HTTP/1.1 extensions for the cache-control of the web documents, gives the ability to the back-end application programs to specify how the dynamic content they generate must be cached or delivered from the proxy.

The majority of previously reported front-end caching approaches handle only the data content that dynamic web pages include. Given that the generation process of a dynamic web page is performed by application programs, the existence of an edge application server communicating with the edge database cache is a prerequisite for the implementation of these approaches. This results to low scalability and transparency of the front-end caching point, and therefore attention should be given to approaches that are based on caching of the whole generation process of dynamic web pages. This is exactly the focus of our proposal, the DOMProxy approach.

3. The DOMProxy Approach

DOMProxy is a proxy-caching architecture for dynamic web pages. The key feature of DOMProxy is the ability to cache the generation process of dynamic web pages instead of the pages themselves. Caching the generation process of a dynamic web page amounts to caching the components that compose it. These components are: (a) the application programs that create the dynamic web pages, (b) the database queries that are generated during applications’ execution, (c) the back-end database content, and (d) the query results that back-end database system (DBMS) produces.

Our proposed architecture deals only with caching of the applications and the query results that are sent as a response from the back-end DBMS. These two different types of web objects are sufficient for enabling the proxy (the front-end caching system in general) to reconstruct and execute the generation process of a dynamic web page. In particular, we are addressing the problem of caching application logic and data content as follows:

Application Programs. Caching application programs is necessary for producing dynamic web pages on request. Hence, the proxy should have some knowledge of the application logic in order to be able to execute the cached applications. To achieve this, we
follow a similar approach like the one proposed in [5], i.e. Cache Applets. As the server responds, we attach all the information the proxy needs in order to fetch the application programs to the dynamic web pages these programs produce.

Query Results. The generation process of a dynamic web page includes three database-related types of objects: (a) the back-end database, (b) the queries performed by the applications, and (c) the query results that the back-end database creates. Hence, the important question that arises is which of these objects should be cached. There are two approaches that can be followed regarding caching of database-related objects. The first one includes caching and manipulating the query results that application programs use. This approach is suitable for creating a more scalable and transparent caching architecture but with confined abilities as far as database content manipulation is concerned. The second approach suggests the usage of full-fledged database caching techniques (e.g. database replication). This approach creates more powerful and efficient database caching architectures; however, it depends strongly on the back-end database and is therefore limited in terms of scalability and transparency. Due to the fact that a proxy must be as scalable and as independent as possible DOMProxy uses the first approach of query result caching.

We proceed now to describe the details of the DOMProxy architecture and its modules. As mentioned earlier, DOMProxy is composed of two main components: a standard proxy that cares for the caching of static content and the Dynamic Object Module (DOM) that provides the functionality for caching dynamic content (see Figure 1). DOM consists of six main components presented bellow.

Interface Manager. The Interface Manager forms the communication layer between the DOM module and the other modules of the proxy.

Dynamic Objects Directory. DOMProxy should maintain the associations between the applications and the corresponding dynamic web pages they produce. The information needed for such a mapping is stored in the Dynamic Objects Directory (DOD). Specifically, DOD maintains (a) the name of the application, (b) the pattern that web pages’ URL should follow in order to be served by this application, (c) the back-end URL of the original application, and (d) the back-end URL of the Data Provider. The Dynamic Objects Directory is also responsible for deciding if an HTTP request that corresponds to a dynamic web page can be satisfied by the DOM. This is accomplished by checking if a requested URL satisfies any of the available URL patterns that the applications can serve. If the needed application to serve the requested URL is cached then the generation process of the corresponding dynamic web page starts with the use of the Application Manager.

Application Manager. The Application Manager is responsible for the manipulation and execution of the application objects that are cached in the proxy. Additionally, it decides whether the DOM module should fetch from the back-end site infrastructure a needed application that is not cached in the proxy. Fetching is done through the Network Manager, which is described below.

Data Manager. The Data Manager controls the storage and manipulation of the cached data objects. The kernel of the Data Manager is an implementation of Semantic Data Caching [8]. Each data object represents a single query, the unit with the lowest granularity in our data manipulation approach. Similar queries are grouped in query classes, which are represented by different semantic spaces depending on their FROM clause, meaning that simple queries over the same base table are stored in the same semantic space. Likewise, in case of join queries, semantic spaces are defined by the set of base tables in the FROM clause and the join conditions.

Each semantic space is n-dimensional, where dimensions are defined from the attributes that appear in the WHERE clause of the corresponding query class. In the general case, tuples of a query result are points in this n-dimensional space; therefore input queries are represented by sets of regions. Due to the fact that the majority of the queries served in web applications nowadays are simple Select-Project-Join (SPJ) queries without negation or disjunctions, in our implementation we process only Conjunctive Normal Form (CNF) queries without negation. An example of a CNF clause without negation is “\((A>5) \land (A<23) \land (B>8)\)”, from which we can see that each query
corresponds to a single continuous region in this n-dimensional space.

The Data Manager’s caching algorithm [8] treats as first class citizens the input queries (semantic regions in our model) and not the base tables. An input query can be answered from the data manager only if it belongs to a family of queries that corresponds to an already cached semantic space. In order to check if a new query is already cached in the corresponding semantic space, the Data Manager creates the semantic region that describes it and checks if it is contained in the already cached semantic regions. For the containment test, all the information that is needed can be acquired from the attributes that correspond to dimensions of the semantic space. Finally, for each semantic space, only the attributes selected from the various queries that arrive to the proxy are cached, i.e. not all base schema attributes.

For example, let’s assume that we have a class of join queries over tables R and S. Two example queries in this class are the following:

\[
\begin{align*}
\text{SELECT} & \quad R.A, R.B, S.D \\
\text{FROM} & \quad R, S \\
\text{WHERE} & \quad (R.FK = S.PK) \text{ AND } (R.A > 5) \\
\end{align*}
\]

These two queries are represented by the same semantic space, defined by the join of R, S over R.FK = S.PK. This semantic space will have a single dimension that corresponds to attribute R.A. Tuples will be stored in a cached table with schema \{R.A, R.B, S.D, S.E\}.

When a new query arrives, the Data Manager checks if the semantic space that this query belongs to is already cached. If this is the case, the Data Manager tries to define the part of the input query that can be answered from already cached data. Following the definitions of [8], let Q denote the constraint formula (WHERE clause) of the given query, and V the DNF clause describing the semantic regions already cached. The given query can be split into two disjoint pieces: the probe query \( P = Q \land \neg V \) that can be answered directly from the cached data and the remainder query \( R = Q \land V \) that must be answered by the back-end server (see Figure 2). After computing the probe and remainder queries, a satisfiability test of R indicates if a query must be posed to the backend server. If R is satisfiable, then R is sent to the backend server and the result of \( P \cup R \) is returned to the proxy as a response. Also, the semantic region that describes R and its corresponding tuples is added to the corresponding semantic space and cache table respectively.

Network Manager. DOMProxy’s caching approach requires the fetching of the needed applications and database-content from the back-end site’s infrastructure. This is done through the Network Manager, which requests the needed objects as they are defined from the Application Manager and Data Manager.

Replacement and Consistency Manager. This component is responsible for defining the replacement and consistency policy for the DOM component.

4. Open Issues

This section discusses other issues related to proxy caching for dynamic content. These issues arise because DOMProxy is based on a group-oriented caching approach, i.e. cached objects are categorized according to their type or their granularity level. This key feature of our approach raises interesting challenges and issues concerning cache’s replacement policy [10], data object caching and the design of cooperative proxy schemes.

4.1. Data Object Caching

The Data Manager caches only the data that are necessary for answering the requested queries. There are cases where groups of these queries present high semantic locality, hence creating frequently requested semantic spaces with high probability to be accessed in the near future. For such areas of semantic spaces we could follow a prefetching strategy by “filling” them with their corresponding back-end data without waiting for a request first. This way, we can create more complete semantic spaces, leading to higher cache hit rates.

Another interesting research issue is that of adding or eliminating dimensions of semantic spaces as the workload changes dynamically. The dynamic change of semantic spaces’ dimensions comes with a considerable cost to the overall performance of our approach. Nevertheless, combined with an algorithm for estimating the dimensions that each semantic space
would have in the future, it might increase the efficiency of our data caching approach.

4.2. Replacement Policy

DOMProxy’s cached objects are characterized from different levels of granularity. Specifically, regarding data-objects, three different levels of granularity are identified as shown in Figure 3. The semantic regions and their corresponding cached data define the lowest granularity level. The corresponding data are grouped according to their semantic description and compose semantic spaces, which comprise the second level of granularity (as explained earlier in Section 3). The different databases that these semantic spaces correspond to, belong to the third level of granularity, databases.

Data objects that are characterized with different levels of granularity present particular characteristics concerning their size, frequency of use, retrieval cost, and temporal and spatial locality; temporal locality refers to the use of the same regions in the data space over consequent accesses, while spatial locality refers to accessing data that is similar in the data space. Thus, for a proxy architecture like DOMProxy, a customizable replacement approach with dynamic granularity would be more appropriate. To be more specific, the replacement policy must be able to change dynamically its granularity and therefore its eviction-criteria according to workload characteristics and the cache storage capacity. This approach presupposes the existence of a set of criteria according to which the replacement granularity and eviction-criteria will change.

There should be three different groups of eviction-criteria, corresponding to the three caching granularity levels. As far as semantic regions and semantic spaces are concerned, their eviction from the cache should be based, among others, on their spatial and temporal locality. We believe that a weight-based algorithm that will take under consideration these parameters, along with standard ones like size and retrieval cost will be the most suitable solution. On the other hand, at the third level of granularity that concerns whole databases, a standard and simple replacement policy like LRU or LFU will be more suitable. The details and alternatives of replacement policy schemes is a subject of our current research.

4.3. Cooperative Proxy Caching

Cooperative proxy caching is defined as the operation of individual proxies sharing their cached objects with each other’s clients [11]. In general, it is used for high client population and geographical dispersion. A factor that affects the efficiency of such a scheme is the cached objects’ distribution among the different proxies. This distribution can be done based on various parameters including cached pages URL, popularity and others.

As mentioned, DOMProxy uses a grouping-oriented caching approach based either to the different nature of the cached objects (application and data objects) or to their semantic description (semantic regions, semantic sets, databases). Moreover, it has the ability to define the existent dependencies among the cached objects regardless of their granularity level. These characteristics can be efficiently exploited for the architecture definition of the cooperative scheme and for the cached objects’ distribution methodology. For instance, additionally to other factors, a distribution method can be based also on their spatial locality. This can be done either at the level of ‘semantic regions’ or at level of ‘semantic sets’. Alternatively, data objects can be grouped, hence distributed, based on their corresponding back-end database.

5. Conclusions and Future Work

In this paper we proposed DOMProxy, a proxy framework that enables dynamic web pages front-end caching. DOMProxy’s caching approach is based on caching the generation process of dynamic web pages and not the pages themselves. The implementation of our approach is based on the front-end caching of the applications and the database content that is used for the creation of the web pages. Furthermore, we
discussed issues and new challenges that are raised by our approach.

Our current work focuses on devising and implementing (a) replacement policies based on group-oriented caching, (b) efficient query containment algorithms, and (c) prefetching algorithms. Furthermore, we are working on the design and implementation of efficient cooperative proxy schemes consisting of several DOMProxies serving a network of clients and servers.

REFERENCES