CFP Taxonomy of the Approaches for Dynamic Web Content Acceleration

Stavros Papastavrou[1], George Samaras[1], Paraskevas Evripidou[1], and Panos K. Chrysanthis[2]

[1] University of Cyprus, Department of Computer Science, P.O.Box.20537, CY-1678 Nicosia, Cyprus
{stavrosp, cssamara, skevos}@ucy.ac.cy
[2] University of Pittsburgh, Department of Computer Science, Pittsburgh, PA 15260, USA
panos@cs.pitt.edu

Abstract. Approximately a decade since it was first introduced, dynamic Web content technology has been gaining in popularity over static means for content dissemination. Its rising demand for computational and network resources has driven researchers into developing a plethora of approaches toward efficient content generation and delivery. Motivated by the lack of a comprehensive study on this research area, we introduce a novel research-charting, semi-formal framework called the CFP Framework, on which we survey and compare past and present approaches for dynamic Web content acceleration. Our framework not only serves as a reference map for researchers toward understanding the evolution of research on this particular area, but also reveals research trends toward developing the next generation of dynamic Web content middlewares.

1 Introduction

The Internet and the Web have become commonplace and their growth is unprecedented. This led to a proliferation of technologies to improve its usefulness and user satisfaction. Dynamic Web content (DWC) technologies facilitate (a) the adaptation of content served to a particular group of people (i.e., people that live within a certain time zone or Internet domain region), and (b) the personalization of content to meet an individual's expectations and needs (i.e., Web banking or e-commerce). According to [13], the above two categories of DWC comprise the 20% of Internet traffic each. DWC enables a new order of Web applications including online financial-related services, news sites, portals and e-learning brokerage platforms.

DWC technology involves a variety of cooperating components that are largely defined as content middleware systems. Arranged in an n-tier architecture, as seen in Figure 1, they cooperate with the goal of delivering content on demand to Web users. A Proxy server intercepts client requests to Web servers and delivers cached content, if certain criteria hold. Otherwise, the request is forwarded to

J. Eder et al. (Eds.): ADBIS 2005, LNCS 3631, pp. 365–378, 2005.

the Web server that invokes the appropriate application server, which generates the content by querying local and/or remote databases.

A dynamic Web page (DWP) consists of static and dynamic content junks called *fragments*, typically arranged in a *template file* interleaved with static HTML code. Dynamic fragments reside in templates in the form of script code blocks that must be processed by the application server. This processing, as illustrated in Figure 1, may require the execution of a significant number of script lines for performing tasks such as database queries, image processing, complex input Form generation, or even information retrieval and manipulation from remote hosts. For instance, the template file of the PC customization dynamic Web page of the www.higrade.com online computer retailer contains approximately 2000 lines of script code, having more than 20 database queries, distributed across 6 dynamic fragments.

Since the generation and delivery of dynamic Web content requires increased computational and network resources, especially during peak hours, various bottlenecks occur. The study in [5] identifies the bottlenecks on three typical dynamic content Web applications. A significant number of research approaches for accelerating DWC are proposed in the literature. As a result, state-of-the-art content middleware technology is found today in many commercial products that incorporate many of the proposed approaches, proving in this way the importance and applicability of this particular research area.

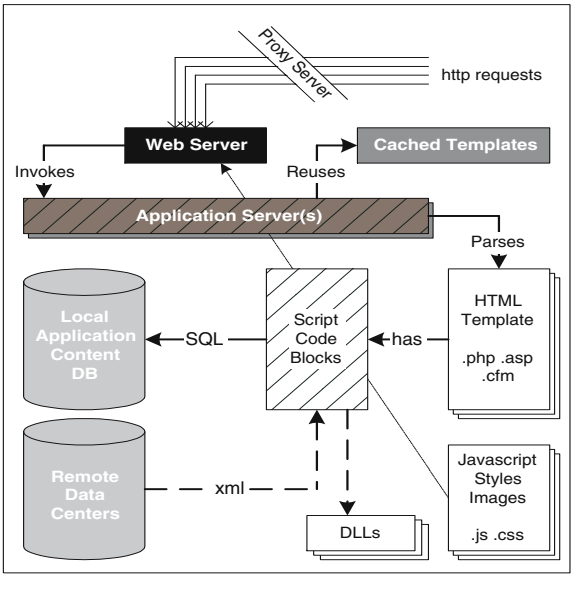

Fig. 1. The n-tier architecture and the process of generating dynamic Web Content

Motivated by the lack of a comprehensive study on this research area, we introduce the CFP Framework, a novel semiformal framework that facilitates the

classification of existing research approaches based on their underlying methodologies and principles. We then attempt a complete literature review and conceptual comparison of the surveyed approaches on the CFP Framework. The purpose of the framework is not to reveal the 'best' approach, but to be a handy tool for researches toward understanding the evolution of research around DWC acceleration. The framework also reveals research trends that can guide researches into defining the next generation of dynamic Web content middlewares.

The next section introduces the CFP Framework and explains the reasoning behind its metrics. Section 3 surveys research on dynamic Web content acceleration and classifies the approaches. In Section 4, we gather and compare the surveyed approaches on the CFP Framework. We recap in Section 5.

2 The CFP Framework

Since a quantitative comparison (i.e., a performance comparison) between the approaches that we survey in the next section is rather unfeasible due to realistic and complexity constrains, we focused instead on establishing a theoretical, comparative semi-formal framework. We consider the fact that the majority of the surveyed approaches employ and combine to some extend three common characteristics or practices. Those are Caching, Fragmentation and Polymorphism and comprise the three principles of the CFP Framework.

The principle of Caching suggests a multi-tier reuse of content on network sites such as proxies, Web servers, application servers, or even at the client browser. The principle of Fragmentation suggests the breaking of a dynamic Web page down to computationally, but not necessarily semantically, distinct parts. This principle enables (a) finer-grained Caching and (b) concurrency in DWC generation. The notion of Polymorphism allows for a dynamic Web page to be assembled in more than one way without the need to regenerate any content. More specifically, the layout of the DWC fragments is decided dynamically according to, for example, the client's preferences. Polymorphism, in this way, enables another dimension of content dynamism by allowing the templates to be dynamic themselves. In the CFP acronym, Caching precedes the other two principles as the earlier to appear, and Polymorphism is the most recent.

The intuition behind the use of the CFP Framework is that the three principles of the framework can be viewed as orthogonal dimensions along which different research approaches can be classified. Thus, the framework can be represented as a cube as shown in Figure 2.

We make use of the CFP Framework by plotting a particular approach on the cube, given its corresponding values for each principle. For that purpose, we define value to be the *extend of employment* of a particular principle. Since the three principles are rather qualitative and subjective than quantitative, this evaluation requires assumptions and approximations in order to define the appropriate metric for each principle. For readability, Figure 2 illustrates the basic metrics only for the principles of Caching and Fragmentation.

The metric for the principle of Caching is the proximity of cached documents. Therefore, we state that an approach that supports Caching of dynamic

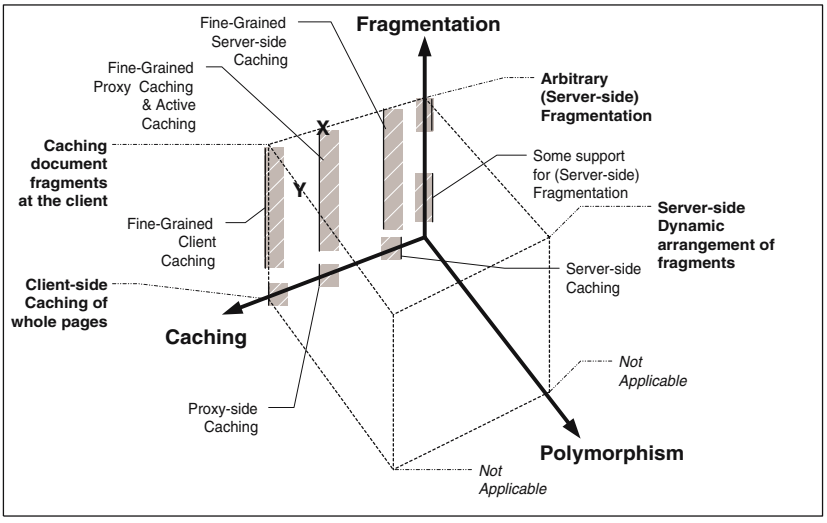

Fig. 2. The CFP Framework and its Approximate Metrics

content closer to Web users is evaluated higher from others that cache content closer to the Web server. Moreover, we state that an approach is fully employing Fragmentation if it supports for an arbitrary number of fragments in a dynamic Web page and of any computational type and size. Finally, an approach fully employs Polymorphism if it provides support for an arbitrary number of alternative arrangements for a dynamic Web page. Since we realize the notion of Polymorphism in combination with Fragmentation, we assert that it cannot be employed as a stand-alone principle or in combination only with Caching. Therefore, it appears as 'not applicable' at the corresponding edges of the CFP framework.

For example, the approach 'X' plotted on the framework in Figure 2 refers to an approach that (a) supports proxy-side Caching of DWC, and (b) fully supports Fragmentation. In another example, approach 'Y' caches arbitrary DWC fragments at the client-side and provides some basic support for different arrangements of the fragments.

3 Taxonomy of Approaches

In this Section, we survey the proposed approaches for accelerating the generation and delivery of dynamic Web content. We present the taxonomies in a more natural and reader-friendly way, rather than applying a strict technical order. Due to the lack of space, we exclude early, assorted and hardware techniques not directly related to the CFP framework.

3.1 Server-Side Fragmentation

The first class of approaches that we survey relate to the principle of Fragmentation. An early form of Fragmentation is encountered in Server-Side Includes

(SSI) [2]. According to SSI, some simple dynamic parts of a page can be isolated and regenerated every time the page is requested such as a counter, the time at the server, and date last modified. [8] suggests a more general form a Fragmentation that allows the dissection of a dynamic page into distinct parts (fragments) that are assembled according to a template file. A fresh version of a fragment is generated every time its underlying data objects are modified, using database triggers. With its fresh fragments in place, a dynamic page can be either immediately delivered or cached (as discussed next). More recently, [21] proposes a technique for accelerating template parsing and execution by processing the dynamic fragments of the template in a concurrent fashion. This approach, however, achieves increased server throughput and lower client response times when the system is not fully loaded. It is worth mentioning that the identification of the fragments takes place at run time (during parsing) and requires no a-priori compilation or special handling of the template.

3.2 Server-Side Caching

Content Caching boosts dynamic content generation by eliminating redundant server load. There are many interesting approaches for server-side Caching that vary mostly on the granularity and level of Caching. In [16] and [14], the Caching of dynamic documents at the granularity of a page is proposed for early content middlewares such as CGI, FastCGI, ISAPI and NSAPI. Extending their work in [14], the authors in [24] propose a Dynamic Content Caching Protocol (DCCP) that can be implemented as an extension to HTTP. This protocol allows for content middlewares, such CGI and Java Servlets, to specify full or partial equivalence amongst different URIs (HTTP GET requests). The equivalence information is inserted by the content middleware into the HTTP response header of a dynamically generated page, and stored at the Caching module along with the cached page. For example, the URI http://www.server.com/LADriveTo.php ?DestCity=newyork instructs the content middleware to generate a page with driving directions from Los Angeles to New York. Prior to transmitting the result page, the middleware inserts the "cache-control:equivalent_result=Dest=queens" attribute in the HTTP response header. The Caching module will cache the page, transmit it to the client, and store the cache-control directive for future use. A subsequent client request for the same URL, but for a different DestinationCity value, will be evaluated by the cache module for a possible mach with the value of "queens" or "newyork". If a much is found, then the cached page is transmitted to the client.

3.3 Fine-Grained Server-Side Caching

To achieve greater reuse of cached content, across both time and multiple users, Caching at finer granularities is proposed. The authors in [28] suggest the Caching of static HTML fragments, XML fragments and database query results. This approach, however, applies to Web applications that follow a strict declarative definition and follow a certain implementation. In addition, Caching cannot be

applied to arbitrary parts of a DWP. A more general, flexible, and easier-to-use method for fragment Caching is introduced later on in [10] and studied more thoroughly in [9]. According to this method, Caching can be applied to an arbitrary fragment of a template by first wrapping it around with the appropriate tags (explicit tagging). XCache [3], is a commercial product that installs as a plug-in on popular dynamic content middlewares, and supports fragment Caching of any type using explicit tagging. Also, the Cold Fusion content middleware provides tags for explicitly defining the fragment to be cached.

3.4 Caching at the Proxy Server

Proxy Caching is the most popular approach for faster delivery of reusable static content such as static HTML pages and media files [26]. A Proxy degrades bandwidth consumption by eliminating unnecessary traffic between clients and servers, given that it is strategically located[1] between them. Proxy servers are found in many network points along the client/server path, with most popular those that reside on an enterprise's network boundaries. It has been identified that the usual hit ratio for proxy caches is around 40% [27], while another 40% of the traffic is redundant when proxies are employed [25].

A popular approach for Web sites to meet the growing demand on DWC delivery is to lease cache space on a service-based network of interconnected proxy servers called Content Distribution Networks (CDN). A typical CDN employs a set of proxy servers strategically arranged by geographical, or network location. Client requests for content are routed to the closest proxy server of the CDN network. The list of popular CDNs includes brand names such as Akamai, Yahoo, Intel and Nortel. It is noteworthy that for a Web site to be registered and served by a CDN network, an off-line procedure of tagging the source code (HTML script files) of the Web site is required. A thorough survey on the procedures, practices and performance of CDNs can be found in [17].

Despite the location of cached content, Server and Proxy Caching find their major implementation difference on how data consistency between the cached content and the underlying database objects is enforced. For the former, cache consistency is easier to be enforced since the Caching module is local to the content middleware (as seen in [8]). For the latter case, efficient cache invalidation techniques are required as discussed later on.

With the evolution of dynamic content middlewares, proxy caches had to adapt by providing support for dynamic documents. Early research conducted in [24], proposes the Caching of dynamic content at the granularity of a page by using the Dynamic Content Caching Protocol as previously discussed. The Caching protocol is applicable for both server-side and proxy-based Caching, and works by allowing the manipulating of HTTP header information and URL query string parameters (GET variables).

Another interesting approach for Caching dynamic pages is found in [19]. Analogous to the Caching protocol approach discussed earlier, this one suggests

[1] *Closer* in terms of network latency, topology or geographic location.

that the proxy server be allowed to examine the HTTP POST variables that are submitted as part of a client HTTP request for a URI . In brief, the proxy server attempts to reuse cached SQL query results by looking up on a predefined mapping called Query Template Info. This mapping establishes a relation between (a) the HTML form fields that are submitted with a URI request, and (b) the SQL query that uses those form fields as query parameters such as WHERE and ORDER BY clauses. Two strong points of this work is that (a) the proxy extracts and reuses portions of cached query results, if necessary, to satisfy future requests, and (b) it compliments a cached query result on demand by negotiating with the content middleware. Since the HTTP post variables are generated from HTML form fields, this approach is called *form-based*. The manipulation of cached content at the proxy server lies under the more general notion of Active Caching as we discussed later on in 3.10.

Both the *form-based* and the *protocol-based* approaches discussed above do not address the important issue of cache consistency. The authors in [6] propose an invalidation technique for cached dynamic pages, which uses a (triple) mapping between (a) the database content, (b) the SQL queries and (c) the dynamical Web pages. This mapping explicitly identifies the database objects that affect a set of queries which in turn are involved in generating a set of DWPs. According to this technique, a cached page is invalidated once a database object that relates to an SQL query which, in turn, is involved in generating that page is updated. Extending their work in [6], the authors in [18] illustrate how this triple-mapping invalidation approach is applied in a real world scenario when all four the database server, the content middleware, the Web server and the proxy cache are entirely independent Vendor products. Additionally, a technique for cached content freshness based on parameters such as user request, database content update rate and network latency is proposed.

3.5 Fine-Grained Proxy Caching

Caching at the granularity of a fragment is consequently proposed for proxy caches. According to fine-grained proxy Caching, the template file is cached at the proxy server whereas its dynamic fragments are either reused from the proxy cache or fetched fresh from the Web server. Edge Side Includes (ESI) was introduced as a standard for Caching page templates along with their fragments on proxy servers [1]. According to ESI, the dynamic fragments of a page are explicitly marked using tag-based macro-commands inside the page's template file. An ESI-compliant proxy server must provide support for parsing the cached template file and executing macros that dictate whether a fragment should also be retrieved from cache, or pulled from the original server. ESI macros have access to a client's HTTP request attributes (cookies, URL string, browser used) in order to choose between fragment alternatives. An example of that would be the identification of the client's browser version or vendor in order to pick the appropriate fragment that meets the browser capabilities. Endorses of the ESI technology are leading CDN brands such as Akamai, database vendors such as Oracle, and content management leader Vignette.

3.6 Client-Side (Fine-Grained) Caching

Surprisingly, the notion of assembling a dynamic page away from the original content middleware was firstly introduced in [12] not for proxy caches, but for client browsers. The proposed technique, called HPP (HTML pre-processing), requires from the client browsers the extra functionality of Caching and processing (parsing) a template file, containing blocks of macro-commands, prior to rendering a dynamic page. Each macro-command block generates from scratch a page fragment by manipulating local variables and strings. This idea can be overview as the client-side equivalent to Server-Side Includes discussed earlier.

Extending their work in [12], the authors in [23] propose the Client-Side Includes (CSI) by merging HPP and ESI. In order to provide support for CSI in the Internet Explorer Web browser, the authors propose a generic downloadable wrapper (plug-in) that uses JavaScript and ActiveX. The wrapper pulls and caches at the client side the template and fragments that are associated with a requested DWP, assembles them together according to the ESI directives in the template, and finally renders the page. According to the authors, CSI is suitable for 'addressing the last mile' along the client-server network path suitable for low-bandwidth dial-up users, even in the absence of an edge server (i.e., a CDN).

3.7 Polymorphism: A Second Dimension of Content Dynamism

Caching at the fragment level requires the existence of a page layout/template that dictates a strict arrangement for cached DWC fragments. If we loose up this restriction, by allowing for more than one template per dynamic page, we achieve Polymorphism (in Greek: the ability for something to show different phases-morphs) in DWC Caching. It is, therefore, left to the content middleware to pick the right template, according to the user's preferences (e.g., the Yahoo! Web site). A recent study in [11] proposes the use of multiple templates for a specific dynamic page along with proxy-cached page fragments. Following a client's request for a dynamic page (e.g., www.server.com/page1.php?id=2), the proxy server always routes the request to the origin content server and causes the execution of the original script (for example homepage.php). This routing is necessary for determining the desired template for page1.php at run time. The selected template is then pushed to the proxy server where it is parsed for identifying which fragments are reused from cache and which ones are requested fresh from the server. After all the fragments are inserted into the template, the complete assembled page is transmitted to the client. The performance analysis conducted in [11] demonstrated solid bandwidth reductions when applying fragment Caching, however, performance analysis for other critical metrics, such as scalability and responsiveness, remains to be seen. We believe that both the necessary routing of each request to the origin content server and the invocation of the original script can heart client response time and server scalability respectively. Nevertheless, the techniques introduced in this approach are an excellent starting point for further research. Polymorphism is also supported by the ESI technology in an indirect manner. Instead of choosing from a pool of templates, basic ESI branching commands may reorganize the layout inside a template according to client credentials (HTTP request header information).

3.8 Support for Inter-dependent Fragments

Caching of dynamic content at the fragment level, as employed today by proxy servers, assumes that individual page fragments have independent Caching characteristics. This assumption has simplified the design and deployment of proxy caches based on either the Edge Side Includes or other proposals. There exist, however, Web applications for which two or more fragments of the same dynamic page are dependent to each other. An example is an online retailer's Web page that includes among others (a) a fragment containing script code for evaluating and regenerating an HTML form used for product customization, and (b) a fragment with script code for calculating and rendering the shopping cart's total charge (including the value of the product being customized by the previous fragment). Upon client submission, the second fragment cannot execute to calculate the total charge unless the previous fragment has evaluated the HTML form input. In this case, the two fragments must execute in a sequential (serial) manner to ensure consistency between the total charge and the customized product's value.

The notion of fragment dependency in dynamic content generation has been studied first in [8] and later on in [21]. In brief, the former study suggests the construction of a separate Object Dependency Graph (ODG) where the objects in this case are the fragments ids (generated fragments are stored as separate files). Furthermore, a dynamic page is defined by a template file with references to fragment files using include statements. In addition, database triggers are installed to ensure that upon database content update, the affected fragments (and their dependent ones) are regenerated in the right order. Client requests for a particular page are fulfilled by inserting into the appropriate template the already generated fragment files. Although the whole procedure requires a quite complex setup, this approach is suitable for publishing heavily requested portals and news sites with frequently updated content and less client interaction. The latter study, suggests a simpler and more general approach for identifying dependencies between page fragments. Instead of using an external fragment dependency graph in conjunction with database triggers, the fragment dependencies are defined at the beginning of each template file. For example, the tag <dependency source_fragment=3 target_fragment=5> informs the content middleware that the third fragment to be encountered during parsing must be executed before the fifth fragment. This inline and immediate definition of fragment dependencies ensures consistency between dependent fragments, since this approach attempts to execute all the fragments of a template in a concurrent fashion (see Section 3.1). As opposed to the former approach, this one is suitable for more interactive Web applications i.e., an online retailer shop.

3.9 Caching with Delta Encoding

Delta encoding is a popular technique for efficiently compressing a file relatively to another one called the 'base' file [15]. This is achieved by computing and storing the difference between the file being compressed and the base file.

Streaming media compression, displaying differences between files (the UNIX diff command) and backing-up data are common applications of delta encoding. Under the assumption that consecutive client requests for a specific URI would generate a sequence of moderately different dynamic pages, Delta encoding can be exploited as an alternative for Caching dynamic content. [22] proposes the Caching of a base file for each group (also called Class) of correlated documents i.e., pages that share a common layout. With the base file cached, the next client request would force the content middleware to compute the Delta between the new dynamic page that the client would normally receive and the base file. The computed Delta is then transmitted from the content middleware to the side where the base file is cached for computation of the new dynamic page. Eventually, the result is transmitted to the client. An interesting feature of this 'class-based delta-encoding' approach, is that the base file can be cached either at the server-side, proxy-side, or even at the client browser itself as long as the required infrastructure exists. In the latter case, Delta encoding benefits could low-bandwidth users. [22] demonstrated solid bandwidth savings and reduced client perceived latency, however, those performance gains reduce the average system throughput to 75% due to increase the CPU overhead of computing the deltas. Nevertheless, we consider Delta encoding for Caching DWC as an exciting open topic of research.

3.10 Active Caching

The notion of Active Caching refers to the ability possessed by a Caching middleware to manipulate cached content instead of requesting fresh content from the server. The approach found in [7] piggybacks a Java object into a dynamically generated document, which is then cached at the proxy. The proxy provides a Java runtime environment in which that object executes in order to modify the dynamic parts of the cached document according to a client's preferences. Examples of document modifications include advertising banner rotation, logging user requests, Server Side Includes execution and even Delta compression. Besides these general types of modification, the Java object can personalize cached documents by retrieving personal information from the Application Database at the server side. Data chunks of personal information are kept by the object for future reuse. Building up on this approach, a more general form of DWC Caching with Active Caching is suggested in [20]. This one is very similar to the 'form-based' approach discussed earlier in the sense that the Java object manipulates HTTP post variables (the Form input) for filling the dynamic parts of the cached document. In general, those two Active Caching approaches differ from other approaches that support Caching and Fragmentation since the dynamic parts (or fragments) are not decoupled (stored separately) from the cached document (or template).

Active Caching of this form can be viewed as a means of content middleware migration. Original executable code and data, both parts of the content middleware, can be packed along with a mobile object (called Mobile Agent) that dispatches to a destination where it can better serve the client (i.e., a proxy

server). This ambitious approach aims at alleviating the processing bottleneck from the content middleware, while reducing unnecessary network traffic by employing Caching and proxy-side computation.

Since the notion of Active Caching is supported by mobile code or function calls that accompanies relevant data, then an alternative or indirect form of Active Caching can be achieved by employing Active XML (AXML) [4]. Documents written according to AXML contain data in XML and calls/references to Web Services that fill-in on demand the missing (or dynamic) parts of the documents. In extend, AXML documents can be cached and materialized prior to transmission to Web users.

4 Conceptual Comparison of the Approaches

4.1 General Remarks

In this Section, we plot the surveyed approaches and technologies on the CFP Framework cube (Figure 3). This allows for a high-level comparison of the approaches, as well as for identifying research trends. In addition to the three dimensions/principles of Caching, Fragmentation and Polymorphism, we show how recent an approach is by using a gray scale background. At the heart of the framework, with null values for each principle, we place CGI as the primary dynamic content middleware. An immediate observation from Figure 3 is that the trend in research is toward refining and extending the employment of the three principles while attempting to combine them. In other words, dynamic content tends to be cached closer to the client, at finer granularities, and under different arrangements. This is crucial for modern Web applications that require content personalization and support for low-bandwidth (mobile) users.

4.2 Detailed Comparison and Discussion

As we observe on the CFP Framework, the majority of early research in accelerating DWC has focused on Caching whole pages (or slight variations of it). Page-level Caching does not meet the fine-grained Caching requirements of modern Web applications. However, within this group of page-level Caching approaches the most recent one that employs Delta Encoding appears very promising interms of performance, especially for low bandwidth users such as mobile users. Therefore, we recommend an implementation that supports Fragmentation that would encapsulate the Caching characteristics of modern Web applications.

Active Caching, as introduced in [7] and explored more in [20], combines the advantages of proxy-side Caching while providing some support for Fragmentation. Both approaches do not employ full Fragmentation since the fragments are not decoupled from the template (are not stored separately), and therefore cannot be cached and reused. For the same reasons, we assert that Server Side Includes (SSI), as discussed in Subsection 3.6, provides the same level of Fragmentation. On the other hand, the references to XML services that the Active

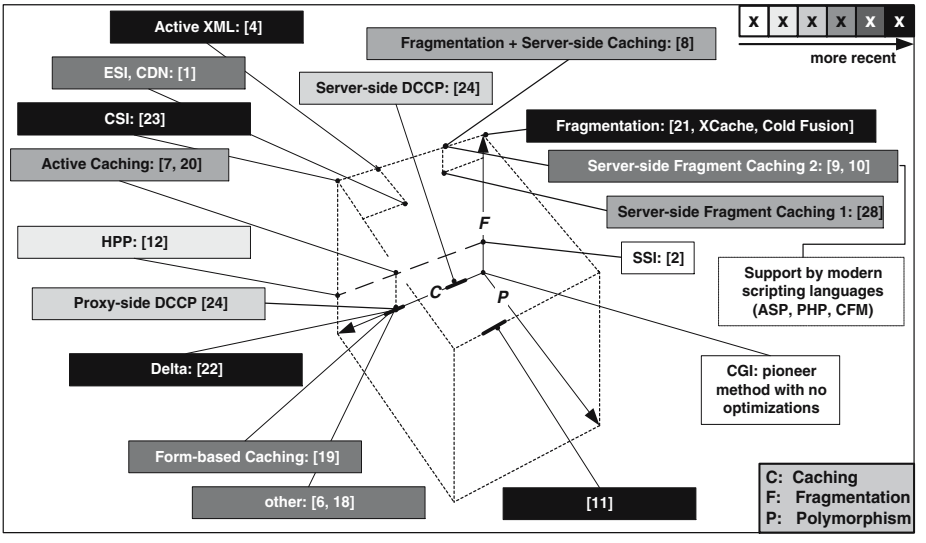

Fig. 3. The CFP Framework with the plotted proposed approaches and technologies. The numbers relate to the reference numbers in the bibliography

XML approach embeds within a template can be reused by other templates allowing in this way for arbitrary Fragmentation.

The early publishing system proposed in [8] supports arbitrary Fragmentation of DWC, however, it provides server-side Caching only at the granularity of page. The recent approach found in [21] supports arbitrary Fragmentation, inner-fragment dependency, and immediate execution of fragments with no Caching. The former approach is more suitable for less interactive Web applications such as portals and news sites since the generation of content is data-driven (i.e., triggered by database changes). The later approach better suits interactive Web applications, such as e-commerce, where fragment generation is user-driven.

The approach proposed in [28] provides server-side Caching but does not employ full Fragmentation. This is because even though the fragments are decoupled from the template, the approach allows for only specific forms of content (such as XML and queries) to be isolated and cached. The more recent approach found in [10] and [9] works around this problem by providing support for Caching of any type of content, at any granularity, on the server. To the same extend, scripting languages such as PHP, Cold Fusion, ASP and XCache provide programming-level support for arbitrary server-side Caching. Edge Side Includes extends [10] and [9] by moving arbitrary fragment Caching from servers to proxies. Finally, [11] compliments ESI (with Polymorphism) by supporting dynamic arrangements of the cached fragments at the proxy. The original Client Side Includes approach, as proposed in [12], employs full Caching, and it targets low-bandwidth clients. However, for the same reasons discussed in Active Caching and Server Side Includes, the approach does not provide full Fragmentation since

it does not allow for arbitrary fragment Caching. The more recent and improved version of Client Side Includes, as proposed in [23], supports full Fragmentation by allowing arbitrary content fragments to be cached at the client browser.

5 Conclusion

In this paper, we surveyed and classified the research approaches and technologies for accelerated dynamic Web content generation and delivery. In order to perform a structured conceptual comparison of the approaches, we introduced the CFP Framework. We believe that our work can be used by researches not only as a study for understanding dynamic Web content technology, but also as point of reference toward developing the next generation of dynamic Web content middlewares.

References

1. The edge-side includes initiative. http://www.esi.org.
2. Server-side includes. http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html.
3. Xcache: The cache management solution. http://www.xcache.com.
4. Abiteboul, S., Benjelloun, O., Manolescu, I., Milo, T., Weber, R.: Active xml: Peer-to-peer data and web services integration. In VLDB (2002)
5. Amza, C., Cecchet, E., Chanda, A., Cox, A., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., Zwaenepoel, W.: Specification and implementation of dynamic web site benchmarks. In IEEE 5th Annual Workshop on Workload Characterization (2002)
6. Candan, K. S., Li, W. S., Luo, Q., Hsiung, W. P., Agrawal, D.: Enabling dynamic content caching for database-driven web sites. In SIGMOD Conference (2001)
7. Cao, P., Zhang, J., Beach, K.: Active cache: caching dynamic contents on the web. In Distributed Systems Engineering 6(1) (1999) 43–50
8. Challenger, J., Iyengar, A., Witting, K., Ferstat, C., Reed, P.: A publishing system for efficiently creating dynamic web content. In INFOCOM (2) (2000) 844–853
9. Datta, A., Dutta, K., Ramamritham, K., Thomas, H. M., VanderMeer, D. E.: Dynamic content acceleration: A caching solution to enable scalable dynamic web page generation. In SIGMOD Conference (2001)
10. Datta, A., Dutta, K., Thomas, H. M., VanderMeer, D. E., Ramamritham, K., Fishman, D.: A comparative study of alternative middle tier caching solutions to support dynamic web content acceleration. In The VLDB Journal (2001) 667–670
11. Datta, A., Dutta, K., Thomas, H. M., VanderMeer, D. E., Suresha, K. Ramamritham, K.: Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. In SIGMOD Conference (2002) 97–108
12. Douglis, F., Haro, A., Rabinovich, M.: HPP: HTML macro-preprocessing to support dynamic document caching. In USENIX Symposium on Internet Technologies and Systems (1997)
13. Feldmann, A., Caceres, R., Douglis, F., Glass, G., Rabinovich, M.: Performance of web proxy caching in heterogeneous bandwidth environments. In INFOCOM (1) (1999) 107–116

14. Holmedahl, V., Smith, B., Yang, T.: Cooperative caching of dynamic content on a distributed web server. In IEEE International Symposium on High Performance Distributed Computing (1998) 243
15. Hunt, J. J., Vo, K. P., Tichy, W. F.: Delta algorithms an empirical analysis. ACM Transactions on Software Engineering and Methodology (1998) 7(2):192–214
16. Iyengar, A., Challenger, J.: Improving web server performance by caching dynamic data. In USENIX Symposium on Internet Technologies and Systems (1997)
17. Krishnamurthy, B., Wills, C. E., Zhang, Y.: On the use and performance of content distribution networks. In Internet Measurement Workshop (2001) 169–182
18. Li, W. S., Candan, K. S., Hsiung, W. P., Po, O., Agrawal, D.: Engineering high performance database-driven e-commerce web sites through dynamic content caching. In EC-Web (2001) 250–259
19. Luo, Q., Naughton, J. F.: Form-based proxy caching for database-backed web sites. In The VLDB Journal (2001) 191–200
20. Luo, Q., Naughton, J. F., Krishnamurthy, R., Cao, P., Li, Y.: Active query caching for database Web servers. (2000) 92–104
21. Papastavrou, S., Samaras, G., Evripidou, P., Chrysanthis, P. K.: Fine-grained parallelism in dynamic web content generation: The parse dispatch and approach. In CoopIS/DOA/ODBASE (2003) 573–588
22. Psounis, K.: Class-based delta-encoding: A scalable scheme for caching dynamic web content. In ICDCS Workshops (2002) 799–805
23. Rabinovich, M., Xiao, Z., Douglis, F., Kalmanek, C. R.: Moving edge-side includes to the real edge - the clients. In USENIX Symposium on Internet Technologies and Systems (2003)
24. Smith, B., Acharya, A., Yang, T., Zhu, H.: Exploiting result equivalence in caching dynamic web content. In USENIX Symposium on Internet Technologies and Systems (1999)
25. Spring, N. T., Wetherall, D.: A protocol-independent technique for eliminating redundant network traffic. In Proceedings of ACM SIGCOMM (2000)
26. Wang, J.: A survey of Web caching schemes for the Internet. ACM Computer Communication Review (1999) 25(9):36–46
27. Wolman, A., Voelker, G. M., Sharma, N., Cardwell, N., Karlin, A. R., Levy, H. M.: On the scale and performance of cooperative web proxy caching. In Symposium on Operating Systems Principles (1999) 16–31
28. Yagoub, K., Florescu, D., Issarny, V., Valduriez, P.: Caching strategies for data-intensive web sites. In The VLDB Journal (2000) 188–199