Journal of Network and Computer Applications ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A survey on dynamic Web content generation and delivery techniques

Jayashree Ravi, Zhifeng Yu, Weisong Shi *

Department of Computer Science, Wayne State University, USA

ARTICLE INFO

Article history:
Received 9 October 2008
Received in revised form
16 March 2009
Accepted 21 March 2009

Keywords:
Dynamic Web content
Content distribution
Personalised Web content

ABSTRACT

While Web applications serve personal needs and business functions almost in every area, the responsiveness and performance of Web applications is the key factor to their success. With continuous innovation on Web technology, Web sites have evolved from document Web to application Web and further to service Web recently. During the evolution course, Web sites serving dynamic content started to grow exponentially to dominate the area. Dynamic pages require servers to generate the response content per-user request before delivering it back to the user, which introduces network traffic, server workload and results in extra latency. This drew tremendous efforts from both research and industry on how to accelerate the dynamic content generation and distribution in order to reduce the user perceived latency and improve the application performance, among which caching is a vital technology. This paper attempts to survey the innovative research and products recently published in this area and presents them in a road map style. It first examines the dynamic characteristics of Web applications and the inherent challenges for caching. Then the rest of this paper explores the varied acceleration solutions on content generation process and content delivery process, respectively, followed by the analysis of how different caching solutions fit Web applications of different characteristics. Finally it ends with the future trends on Web caching technique and a summary of the survey.

1. Introduction

As we are heading into Internet Services Era, ‘‘online’’ becomes a necessity for any software system which attempts to prevail in today’s market. Even the giant software maker Microsoft started to sense the urgency and decided to ‘‘go live,’’ after many Web-based services such as Google Inc and Yahoo Inc were gaining momentum in this area recently. The ‘‘online’’ software (also known as software-as-a-service, SaaS) is enabled by low-cost bandwidth, widespread wireless access, and cheap memory and storage, and marked by rapid release cycles, an agnostic attitude about whether products run on PCs or other computing devices, and the ability to blend local, peer-to-peer, and online functions (Ricadela, 2005). Needless to say, since the last 10 years when the majority of Web sites were developed to publish static information only, they have evolved steadily and ubiquitously to serve dynamic and complex Web contents and business functions.

From the first generation marked by document Web, Web sites are evolving into application Web, and further towards service Web recently. Document Web simply has Web server host Internet accessible documents, most of which are static HTML and images. Most of the traditional content delivery acceleration technologies addressed document Web only. With the surging popularity of Web sites, the application Web aligned with e-Business models started to emerge, which relies on server side programs executing business logics hosted on application servers to generate the dynamic HTML as per each user request, for example, the online book store, auction site, and enterprise Web applications. The Web application growth provides the opportunity of e-Business process integration which in turn pushed the technology advancement and resulted in service Web which is yet to mature. The service Web is powered by application server which supports Web service and delivers the XML as response. Both application Web and service Web require dynamic content generation on server side for each unique request. In this paper, we do not specifically distinguish these three types of Web when discussing about the content generation and delivery and they are all referenced as Web application hereafter.

Responsiveness, i.e., user-perceived latency, is the most important performance measurement which people use to determine if application Web fits personal or business needs. A great amount of efforts from both research and industry sectors have been made to improve the Web performance by reducing the client response latency and network traffic. The work varies from increasing network bandwidth, improving data transmission rate, caching the content in every possible location, to scaling up the server farm by any means.

On the other hand, there is significant increase in on-the-move access to the Web content using a variety of mobile resource

* Corresponding author.
E-mail addresses: jayashreeravi@ymail.com (J. Ravi), zhifeng.yu@wayne.edu (Z. Yu), weisong@wayne.edu (W. Shi).

1084-8045/$ - see front matter © 2009 Published by Elsevier Ltd.
doi:10.1016/j.jnca.2009.03.005

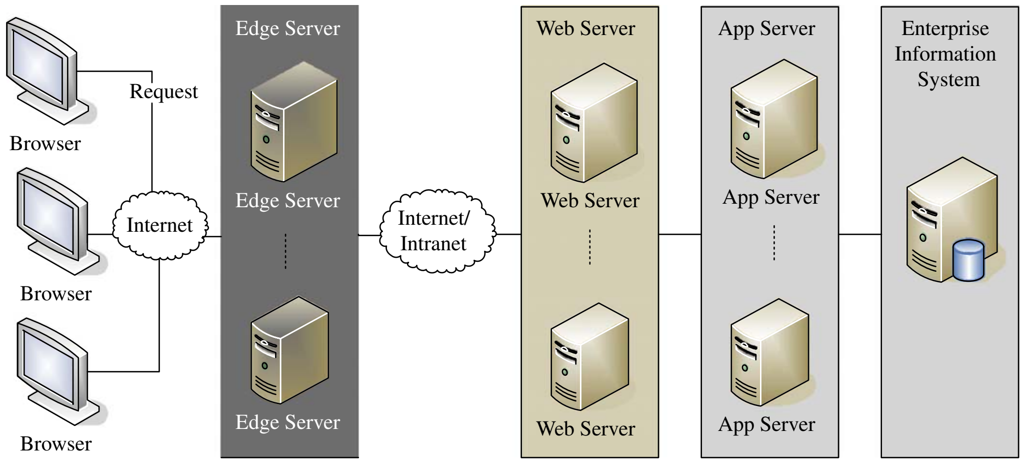

Fig. 1. A typical Web application system model.

constrained devices. It is observed that the Web enabled PDA and cellular phone as well as the efforts in trying to "mobilize" Web content keep growing explosively. As mobile communication cannot be as stable as wired network inherently, the requirements for responsiveness are even higher. All these enforce a demand for the technologies to accelerate the content generation and distribution.

Web application needs a more complex infrastructure than static content Web site, as Fig. 1 shows. This system model is a typical one in practice and will be referenced in the rest of this paper. It consists of

(1) Edge server, which extends the service from the original server to the Internet edge close to user by caching content and application code in order to reduce the client latency and offload the original servers. The new generation edge servers support caching the application code as well. (2) Web server, which takes HTTP request and sends response back to client. (3) Application server, which executes the business logic. The application server normally connects to the back-end enterprise information systems, and performs read or write operation driven by business rules. Most of the dynamic contents are generated on application servers. (4) Enterprise information system, where the content is originally stored. It can be of varied form, for example, XML, file system or DBMS, where DBMS is the most popular information system type in practical use.

With this typical Web application deployment infrastructure, there are many variables which can influence the user perceived latency. This can be notated as follows:

$$L_t = L_{dns} + L_g + L_{nt} + L_{na} + L_u \quad (1)$$

where the total user perceived latency L_t is a sum of the latency introduced by the DNS lookup L_{dns}, latency introduced by the origin server to generate the dynamic page L_g, latency introduced by the network traffic L_{nt}, latency introduced by the Internet access used by the user (dial-up introduces more latency compared to DSL or cable access) L_{na} and latency introduced by the speed of machine the user is using to access the Web pages L_u.

Dynamic content generation and delivery will cost both network bandwidth and response latency if content caching is not possible. Caching serves four main purposes: improving the user perceived latency, reducing the Internet traffic (bandwidth availability), improving the scalability and availability of the origin server. Although the legacy content caching and content distribution network (CDN) improved the static Web site performance significantly, it lacks the proactive and effective invalidation mechanism required for caching dynamic content. The key issues with caching dynamic content is to determine what should be cached, where cache should be placed and how to invalidate the cached data efficiently. These new challenges and issues drew the researches on: (1) what characteristics of Web application bring challenges of caching dynamic content: analyze the cacheability and challenges for dynamic content and study how to improve it; (2) acceleration of dynamic content generation: study how to accelerate the content generation via caching within architecture, on database or edge computing, implement these approaches in adaptive or programmatically; and (3) acceleration of dynamic content delivery: explore new caching strategies and content delivery approaches for dynamic content.

While one relevant survey by Sivasubramanian et al. (2007) reviewed content caching techniques for database-driven Web applications and the other one by Pathan et al. (2008) surveyed the technologies used in content distribution network, this survey extends the scope to the acceleration technologies of dynamic content Web from its generation to delivery and includes caching of both content and computation. The rest of this paper is organized as follows. In Section 2 we analyze the characteristics of Web content. Section 3 explains how the different caching techniques and deployment models help to accelerate content generation. Different types of content delivery approaches are discussed in Section 4. Section 5 further investigates how these content caching solutions are applicable to applications with different characteristics. Finally, in Section 6 we discuss the remaining challenge and research opportunities in this area and summarize our survey in Section 7.

2. Dynamic content Web site characteristics

This section defines the salient characteristics applicable to dynamic pages, which can help us propose good solutions for improving the performance. However, when it comes to characteristics of Web applications, researchers, to start with, define the characteristics based on heuristics. Some of the heuristics are proved to be correct by the same or other researchers at a later point of time. Rabinovich and Spatscheck (2002) defined a set of rules of thumb which are applicable to any Web application. Rules of thumb are generally based on popular real life experiences. Dynamic Web pages form a subset of the Web application. Hence these rules of thumb can be extrapolated and applied to dynamic Web pages as well. The rules defined are: (1) The mean Web object size is of the order of 10–15 kB and the median is 2–4 kB. (2) The popularity of Web objects are very uneven: a small fraction of objects are responsible for a majority of accesses. The popularity distribution of Web objects can be approximated to a Zipf-like distribution. This heuristic is proved correct by research (Breslau et al., 1999). (3) The access rate of objects is typically much higher than the modification rate. (4) On a time scale below 1 min, Web traffic is very bursting. Because of the burst, characteristics averaged over a period of a few tens of seconds or less are in general unreliable. (5) A non-negligible fraction of Web accesses, between 5% and 10% are aborted. Knowing these characteristics helps us design the generation, dissemination and delivery techniques which help us get the best performance. For e.g., knowing that the popularity distribution follows Zipf-like distribution can help us design a caching solution based on Zipf-like distribution which would enable us identify the inactive pages which can be considered for elimination to conserve memory. There are many generation, caching and delivery techniques which are designed to take advantage of these characteristics to improve performance. The same techniques can be applied to dynamic pages as well. Detailing the general solutions applicable to Web applications is beyond the scope of this paper. In addition to the above characteristics which are applicable to Web applications in general but never the less is also applicable to dynamic pages, there are some special characteristics which are applicable to dynamic pages alone. We detail these special characteristics below.

The most significant characteristic of a dynamic page is that dynamic page changes with time more frequently than static pages. Though caching a Web application object is a general requirement, caching a dynamic page which changes very often is a very tricky problem. Simple caching techniques which are employed for static pages would not be an efficient way for caching dynamic pages. Most of the caches and proxies deployed in the earlier times would only cache static pages. Caching of dynamic pages was introduced in the recent past. Proxies and caches which cater to only static pages needed a way of determining whether to cache a response or not based on studying the salient characteristic of a dynamic page. HTTP1.1 provides a clear rule, which is every page is cacheable as default unless the response header indicates "no cache" otherwise. This is based on the assumption that dynamic pages will have a "no cache" tag in the header. However, many dynamic pages are generated without declaring "no cache" in the header. If such pages are cached and displayed to another user at a later point of time the user will get obsolete data. To overcome such problems proxies and caches follow common heuristics to identify a dynamic page by studying some of the subtle differences noticed in the request object, response object and the URL which differentiate them from a static page (Rabinovich and Spatscheck, 2002). Proxies and caches do not cache the following: (1) If the URL contains "cgi-bin", a question mark "?", or a suffix ".cgi". This heuristic is based on the fact that URLs of this form usually identifies dynamic pages. (2) If the request contains a cookie header and response contains a set-cookie header. This is again a heuristic and the rational here is cookies are used to personalize a dynamic page. (3) If the requests use methods other than common "GET" and "HEAD". Zhu et al. (2004) defined seven un-cacheable subtypes based on HTTP header methods, viz., NonGet, DynGet, Pragma, CacheCtl, Personalized, AbnormalStatus and ZeroTTL. If a page contains any of these headers then that page can be safely considered to be a dynamic page. (4) If the request contains authorization header or with "307 Temporary Redirect".

The above differences which identify a dynamic page from a static page would be based on many mechanisms which are specific to dynamic pages such as the generation method used to generate the dynamic page, the type of the data being collected by the Web site from the users, and the type of users it is intended to and so on. If the proxies and caches stop caching based on the above, then about 40% of total requests are un-cacheable (Feldmann et al., 1999; Wolman et al., 1999). However, the research community has proposed caching solutions to this 40% of Web pages, which are explained in detail in Sections 3 and 4.

Types of dynamic pages: We can broadly classify all dynamic pages into two types. The first type which we henceforth call just *dynamic pages* are those that are generated without taking sessions of the user into account. This type does not need to know who has accessed the page and for every user the dynamic page generated is the same at any instant of time. The second type, which we call *personalized dynamic pages* are generated when the user accesses them through a secured system. In this case the dynamic page generated is tailor made for each user. All the major Internet service providers do provide personalized content with iGoogle and My Yahoo! as typical examples. Though both these types of dynamic pages inherit the general characteristics of dynamic pages, they still have certain salient characteristics which distinguish them from each other.

(1) *Characteristics of dynamic pages*: In this type, the same page will be shown to every user who access the Web site. However, the page itself may be changing at a very fast rate making it difficult to cache. For example, Web pages containing real time stock quotation is not cacheable as stock prices may vary every second. However, it was observed that if a page is divided into fragments, some or all of the fragments, for example, the header, footer of the page, become reusable and cacheable. The portion which shows the stock prices would still not be cacheable. This notion further developed into a two popular fragment caching schemas: ESI (edge side include) and DE (delta encoding). While ESI approach divides page into cacheable and un-cacheable fragments, the DE breaks the page into base file and delta fragments. Fragments are assembled on the edge server side closest to the client to minimize the content generation request on original server and therefore reduce the network traffic, the burden on original server side and improve client response time as well. The delta encoding approach is based on the observation that two different versions of the same dynamic Web page share great similarity. Therefore, the common portion of the page can be cacheable and reusable, called "base file", while the other portion has to be generated dynamically, called correspondingly "delta". These are explained in detail in Section 4.

(2) *Characteristics of personalized dynamic pages*: If a new dynamic page has to be generated for each user request then caching of such a page could be impossible. For such Web application, each request is unique even when the URL looks identical, as personalization and customization are the key features of this type of Web application. The HTTP request is represented by the URL with associated parameters, which may be invisible depending on what request method is used. As two most popular methods, "GET" method attaches the associated parameters with the URL in a string of key-value pairs. "POST" method includes the parameters in the request but not visible from the URL itself. HTTP header also contains some user specific attribute and drives for different response content. For example, the "locale = es_ES" indicates the locale and internationalized Web application will display the content in Spanish automatically. In addition, the client cookie is usually carried with request which stores user or session information. Bent et al. (2004) analyzed the trace for 3000 popular Web site and concluded that 47% of workload could not be cacheablesimply because they have a cookie attached and overall 66% of responses across the Web sites are not cacheable, surprisingly higher than previous estimation.

The prevailing use of session object makes almost each request unique and harder to cache the response. Usually session object contains some important attributes, for example, business state, user identity, user authorization and session id, in between requests, which makes one request affect or be affected by the service of another request. These attributes can only be read on original application server unless application code is deployed on edge server and session data consistency is maintained in the infrastructure.

Interestingly research community has proposed several solutions for caching of both dynamic and personalized dynamic pages which are explained in the next section.

3. Dynamic content generation and caching

Most of the Web pages today are generated dynamically on the server side before they are delivered to clients. Even with content caching proxy available, if cache hit misses or the cache is invalid, the content has to be generated on demand. The latency introduced to generate the dynamic page L_g is determined by the performance and scalability of original servers.

Content generation process involves the tasks of executing business logic, accessing data, and generating the presentation, all occurring on server side. To help the future discussion on caching technologies in this section, we use a typical Web application with common model-view-control (MVC) architecture to illustrate the caching options.

As Fig. 2 shows, the controller takes HTTP request, interprets the URL and associated parameters, and maps them into the command to be performed by the model. It also determines which view will be displayed as the response back to the client. The model represents the business data and business logic or operations, which retrieves or computes on the business data usually stored in DBMS system. The view is simply responsible for accessing data from the model, which may have been previously created and cached by the controller, and populating the presentation templates with data.

The MVC architecture clearly separates data access from business logic, presentation from content, and inherently provides the caching opportunity in application design. With regard to where caching can be implemented, Mohan (2001) partitioned J2EE application into four logic tiers: edge of enterprise or network, presentation logic, business logic and data, and caching can be placed in each tier. The latter three tiers correspond to the view, model and data access in the model. The significance of caching in different tiers varies, and the implementation can be provided by application developers, middle ware product vendors, application server vendors and DBMS vendors.

We hereafter elaborate these caching technologies from four aspects: (1) dynamic content caching within infrastructure; (2) database caching strategies; (3) application deployment models to support caching; and (4) programmatic or transparent caching solutions.

3.1. Dynamic content caching within infrastructure

When Web applications tend to be more interactive but lack temporal locality which traditional CDN technology relies on, it becomes a key requirement to accelerate content generation on original servers. More specifically, content caching can be offered by application server vendors, DBMS vendors, or application specific caching implementation in application framework.

Almost all market leading J2EE application servers, namely IBM WebSphere (IBM Corp,), Oracle WebLogic Application Server (ORacle Corp,), provide the dynamic caching features. The built-in cache service is an in-memory cache system to store cached objects and it includes the cache replication service as well. For example, IBM WebSphere application server supports Websphere:01: (1) Servlet/JSP result caching, based on request parameters and attributes, URI and session information. It can be used to cache both page level and fragment level. (2) Command result caching on the Java application level, used to cache the dynamic data that require back-end data retrieval or additional computation or manipulation. (3) Caching replication allows the data to be generated once and then be replicated to the other server in a cluster environment. Cache consistency is also managed by application server. In a clustered server environment,

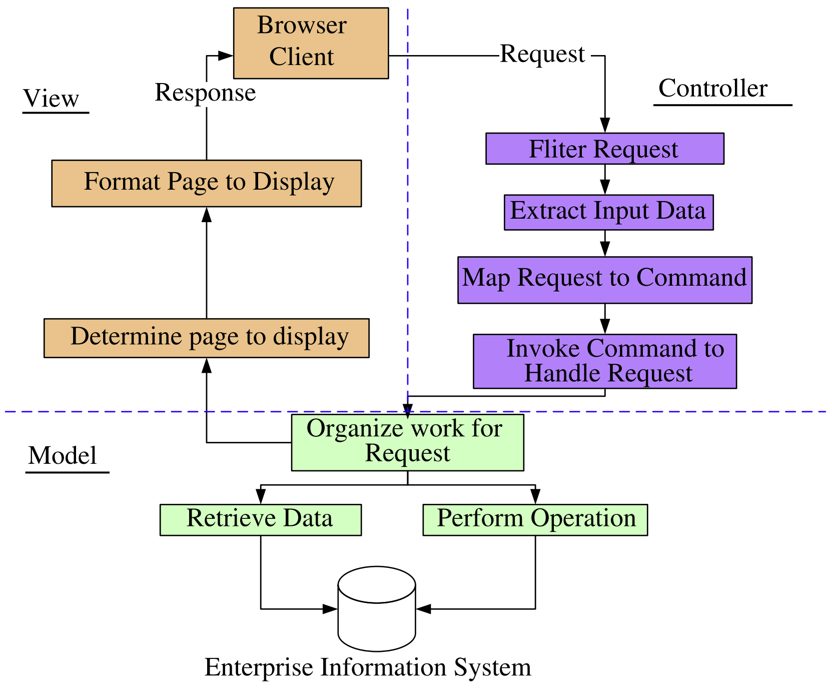

Fig. 2. The model-view-controller architecture (Singh and Stearns, 2002).

if the cache entry is invalidated on one server, the invalidation is propagated to all servers. This caching option does not require code change and it reduces the latency relatively more as cache is maintained in Java heap memory. However, it is limited by the physical capacity of heap memory and programmers have little control over the caching mechanism.

For the Web application with MVC architecture, it is easy to conceive that there are two locations where caching can improve overall performance. One is the view where page can be divided properly into fragments which represent useable Web objects, the other one is the model where the query result can be cached and reused for consequent requests. The popular ESI (Tsimelzon (2000)) technology enables Web pages to be broken down into fragments of differing cacheability profiles. These fragments are maintained as separate elements in the application server's local cache and/or on the content delivery network. ESI page fragments are assembled into HTML pages when requested by end users. This implies the opportunity that much more dynamically generated content can be cached, then assembled and delivered from the edge when requested. Zhu and Yang (2001) proposed a class-based cache management schema, which partitions the page into classes based on URL patterns so that application can specify page identification and data dependence and invoke invalidation for a class of dynamic pages. They further proposed the selective pre-computing to refresh the stale pages and smoothen the load peaks. However, this approach in general only fits into application with relatively slower changing data, rather than the frequently changing data like stock quotations.

Others explore the benefit of caching intermediate query results. Yagoub et al. (2000) discussed what, when and where to cache the data to reuse the intermediate computation results to answer subsequent queries. They also implemented weave management system to cache database data, XML fragments and HTML files. Although the system is build for declarative data-intensive Web sites, the caching policies and the approach to optimize performance apply to most Web applications with MVC architecture and XML/XSLT views. Another dynamic content caching and invalidation framework "CachePortalI" for accelerating database-driven Web application is proposed in Li et al. (2004). The suggested technology enables the dynamic content caching by automatically deriving the relationship between cached pages and database contents (i.e., URL and query mapping), and intelligently monitoring database changes to "eject" stale pages from caches. A case study on the Java Pet Store shows, if the proposed guidelines are followed, more than 90% of the content can be cached and the Web site can be scaled up more than 20 times, and this can be achieved without modification to the underlying application business logic and without sacrificing functionalities.

However, there is a cost to manage cache consistence and cache replacement. If a cached object has to be invalidated and refreshed very often it is not worthy caching. Server side caching object in memory will demand resource and potentially impact application scalability. To manage the caching cost efficiency given limited resource, Mahdavi and Shepherd (2004) introduced the cache-worthiness scores calculated based on parameters of access frequency, update frequency, computation cost and delivery cost. The best candidates for caching are objects who are requested frequently, not changed frequently and expensive to compute or deliver.

3.2. *Database caching strategy*

We list database caching strategy as a separate section, in recognition of its vital role in the Web application, as most of the dynamic content is extracted and generated from back-end databases. To identify the performance bottle neck of the Web applications, the benchmark on some typical e-commerce applications is conducted by Amza et al. (2002) and Liu et al. (2004) independently, results point to database performance as the bottleneck. Researches also find that using dynamic caching can substantially reduce the CPU utilization but not always the number of DISK I/O of the database server. In most of the cases, dynamic caching reduces temporal locality in database page references but to a smaller degree than that reported in file servers and Web proxies.

Recent years have witnessed that more researches move on to the middle tier database caching. Luo et al. (2002) observed that simply cloning Web applications could scale up to heavy workloads but eventually leave the back-end DBMS as performance bottleneck. They initiated and developed research prototype of "DBCache" to enable a regular DB2 UDB instance to become a DBCache without any application modification. Four caching schemas are presented with category of the units of logical data: full table, a subset of a table, an intermediate query result or a final query result. Soundararajan and Amza (2005) proposed the use of semantic information to improve performance of transparent query caching for dynamic content Web sites. The approach avoids the cached query response invalidations what would otherwise occur due to the addition of new records by keeping newly inserted rows in small temporary tables. This allows reusing the cached query for partial coverage of query results. Another transparent caching approach is proposed in Amza et al. (2005), where they start with a coarse-grain table-level automatic invalidation cache and enhance the cache with the necessary dependency tracking and invalidations at the finer granularity of columns based on observed workload characteristics. In addition, the approach reduces the miss penalty of invalidations through full and partial coverage of query results. And the system design could allow query cache be located at the database back-end, on dedicated machines, on the frond-ends or on a combination thereof.

Other middle tier database caching mechanisms work with specific programming language. Hibernate is an object/relational mapping (ORM) approach to provide bridge between relational data and Java object. It provides multiple level caching to cache the Java object of classes which correspond to table rows, in addition to the query result caching. It improves the performance of data access layer when the cached object is rarely changed and local to the application. C-JDBC (Inria,) provides transparent database clustering (partitioning, replication, etc.) to any Java application through JDBC. It presents a single virtual database to the application through JDBC. While its primary purpose is to support database scalability by combination of both vertical and horizontal approaches, the query response caching improves the performance even with single database back-end.

Database cache can be further extended to the edge server for Web application to offload the original server, but caching dynamic data is a great challenge. The general requirements for caching data on edge server are: independence of DBMS; self-management; fast query matching; efficient space management; and consistency management. DBProxy (Amiri et al., 2003) is an edge data caching implementation. It maintains partial but semantically consistent materialized view of previous query result, designed to adapt to the changes in the workload in a transparent and graceful fashion by caching a large number of overlapping and dynamically changing "materialized view". The cache replacement mechanisms address the challenges of shared data across views and adjusts to operate under various space constraints using a cost–benefit-based replacement policy.

When it comes to the database caching schema design, the preassumption about user access pattern and workload is one of the most important design factors and validity of the assumptions determines the actual performance gain. The assumption specifically affects the choice or replica placement. For example, Soundararajan and Amza (2005) assumed that high frequency of browser-type access to newly inserted items as a common application pattern. In Sivasubramanian et al. (2005) caching schema is based on the belief that typical Web application issues more than 80% of simple queries, which use primary key for search and result in exact match. On the other hand, the complex queries base on secondary keys and span multiple tables. This premise drives the design of GlobalDB and its strategy for replica placement. The system directs all simple queries to the closest replica and the complex ones to central server. Another system GlobeTP (Groothuyse et al., 2007) assumes that a Web application's query workload is composed of a small set of read and write templates. Therefore, a partial replication is exploited based on the knowledge of these templates and their respective execution costs. Hibernate works best for reading rarely changed objects. But the concurrency management is very costly and results in noticeable decreased performance and deadlocks on cached objects, when user access pattern moves to massive updating. On the other hand, Sivasubramanian et al. (2005) specifically discussed the data placement strategies for update-intensive Web applications.

Finally, among the researches in this area, some of them are attributed to data consistence management. We will discuss them in the context of edge computation later in this paper.

3.3. *Application distribution model*

Web application has to deal with both of the data and the computation which generates content out of the underlying data. The real challenge is how to distribute both computation/transaction and database globally, and subsequently where to place replicas and how to maintain consistence. Replicating a Web application requires the distribution of both application code and data replica. There have been researches on both aspects: replicating database to reduce the data access latency yet with consistency maintained, and distributing the computation to offload the original server. The key is how to manipulate shared data without incurring the availability and performance penalties that could cost by accessing traditional centralized database.

The industrial market has seen a few commercial products like IBM WebSphere Edge server IBM Corp, Oracle RFID ORacle Corp. Edge server improves response time by offload back-end server computation or data accesses. Akamai Technologies Inc also introduces the EdgeComputing Model (Davis et al., 2004), which is the new deployment model for Web application. To deploy Web application with the edge service product, the developer typically must split the application into two components: an edge component deployed on edge server and an origin component deployed in the traditional manner within the central data center. The most common model for EdgeComputing is to deploy the presentation layer of an application onto edge server and to cache access to original server via the Java Web service client model. Edge server replicates the user session states, permitting the edge components to maintain per-user states that remain available even when the user is mapped to different servers.

Edge service can significantly benefit the Web application either with static database or not requiring the database, by dynamically assembling the content. Yuan et al. (2004) evaluated the different strategies of edge caching and their design/deployment tradeoff. They applied five different application portioning and offloading schemas on the Pet Shop.Net application, and measure their performance, respectively. With the benchmark result, they argued that with great benefits that can be reached in general advanced offloading strategies can be overly complex and even counterproductive. In contract, simple argumentation at proxies to enable fragment caching and page composition achieves most of the benefits without compromising important considerations like security.

In addition, edge service expects the developer to divide the application into edge and origin components, which may require the redesign of the application to gain the most benefit. Gao et al. (2005) demonstrated how the knowledge of application specific semantics can help to maximize the caching effectiveness, where the specific distributed objects are designed to manage a specific subset of shared information using a simple and effective consistency model. It shows that their object-based edge server system provides a factor of five improvements in response time over traditional centralized cluster architecture and a factor of nine improvements over an edge service system that distributes code but remains a centralized database. However, this is achieved by the data replication specifically designed to slightly relax the consistence.

Stronger data consistence always means the higher cost for data replication. This leads to research work to study how to obtain strong consistence and performance at the same time. Different than the work presented in Gao et al. (2005) which achieves availability by relaxing the consistence, Sivasubramanian et al. (2004) attempted to maintain strong consistence with high scalability and further proposed the design for replicating Web application on demand, where data units are replicated only to the servers that access them often. This will reduce the consistency overhead as updates are sent to a reduced number of servers. A middleware layer called Ganesh Tolia and Satyanarayanan (2007) is developed to reduce the volume of data transmitted without semantic interpretation of queries or results. It achieves this reduction through the use of cryptographic hashing to detect similarities with previous results, while not requiring modifications to applications, Web servers, or database servers, and works with closed-source applications and databases.

Cannataro et al. (2002) proposed a new data centric model for generating dynamic pages by proposing an application domain model with XML and object-oriented approach. The model makes use of XML for the description of meta data about basic information fragments. They propose a three-dimensional approach to model different aspects namely, user's behavior, external environment and technology. Based on these aspects a dynamic page is generated.

3.4. *Programmatic or transparent caching solution*

In terms of programming model, those suggested approaches or products for accelerating content generation can be categorized into two types: Programmatic approach and Transparent approach. The former approach is per-application, specific solution requiring either the code change to capitalize the caching options or the site administration tasks, for example, Hibernate requires developer to code with the Hibernate generated classes in stead of the SQL language. The latter one is transparent as it does not require any change on application code or server product, C-JDBC (Inria,) is an example which can implement data replication and query caching without any change on application and database engine except for it works with JDBC applications only.

Programmatic approach takes advantage of semantic knowledge of application code and data, therefore can potentially maximize the performance improvement without comprising other requirements. Gao et al. (2005) demonstrated that given the semantic knowledge of data they can achieve higher availability and efficiency by slightly relaxing the data consistency. ESI (Tsimelzon, 2000) requires application to use ESI scripts to describe cacheable and non-cacheable Web page components. And ESI for Java provides extensions to Java that make it easy to program JSPs using ESI. "CachePortalII" (Li et al., 2004), Weave System (Yagoub et al., 2000), and "query caching" (Soundararajan and Amza, 2005) are other examples of programmatic approach. WebSphere Edge Server (IBM Corp,) and Akamai EdgeComputing Server (Akamai Technologies Inc,) both require developer to partition the origin and edge components and declare them while deploying applications.

On the other hand, the transparent approach is characterized by the automatic replica replacement and invalidation. This requires the system to self-learn the client access pattern and workload in real time and autonomically adjust to the reality. Otherwise, the strong data consistence will sacrifice the system performance and scalability. The research work GlobalDB (Sivasubramanian et al., 2005) and Amza et al. (2005) demonstrated how to maintain both strong consistence and performance yet transparently. DBCache (Luo et al., 2002) is an extension to DB2/UDB database, does not require any programming effort when utilized, neither does C-JDBC (Inria,). And DBProxy (Amiri et al., 2003) is designed to adapt to the changes in the workload in a transparent way.

Plattner and Alonso (2004) proposed Ganymed which uses a scheduler to route transactions to a set of replicas using RSI-PC scheduling algorithm. The key idea behind RSI-PC is the separation of update and read-only transactions. Updates will always be routed to a main replica, whereas the remaining transactions are handled by any of the remaining replicas potentially unlimited, which act as read-only copies. A JDBC driver enables client applications to be connected to the Ganymed scheduler, thereby offering a standardized interface to the system and also helps in keeping the access pattern transparent. The evaluation shows that Ganymed offers almost linear scalability. The scalability manifests itself both as increased throughput and as reduced response times.

In a summary, accelerating content generation remains a big challenge despite various caching technologies proposed or readily available in every phase of content generation process. There is no single technology that fits all scenarios as each Web application is unique from each other. Generally speaking, with user requirements and queries being somewhat predictable, it is common that a customized design will achieve better performance if both cacheability and scalability exploited. For example, Amazon SDB follows its unique "eventual consistency" data model and utilizes its own cloud computing platform EC2 and S3 (Amazon Web Service,) to maximize the performance. But it is a big challenge to predict the user access pattern dynamically and adapt to it in a real time fashion.

On the other hand, some emerging trends in Web application development deserve further research. Firstly, gaining popularity of Web 2.0 applications require more interaction and better responsiveness which contributes to rapid adoption of asynchronous HTTP request–response model. But how the back-end content generation process adapts to this trend is not explored. Secondly, even a well-designed Web application could fail the user expectation when facing unusually overwhelming requests, which has more to do with the scalability of content generation. While cloud computing supports the scalability in a dynamic way, very few research is present on how content generation can be accelerated further by leveraging cloud computing.

4. Dynamic content delivery

This section focuses on the caching techniques for dynamic page delivery. These are the solutions which are applicable between the Web server and the client of Fig. 1. The available solutions can be broadly classified by viewing them in two orthogonal dimensions, viz., based on location of the cache and type of cached object, as shown in Fig. 3.

4.1. *Caching classification based on location of cache*

The techniques for dynamic content caching which are proposed by the research community over the years are equally challenging and interesting. The most popular classification is mainly based on the location of the cache: server side effort (Challenger et al., 1999, 2000; Zhu and Yang, 2001), proxy side effort (a.k.a edge side effort) (Cao et al., 1998; Douglis et al., 1997; Mikhailov and Wills, 2001; Myers et al., 2001; Shi and Karamcheti, 2001), and client side effort (Rabinovich et al., 2003).

In case of server side solutions, caches are placed very close to the origin Web site and for all practical purposes, caches and origin Web sites can be considered to be on one point of the network. Many industry leaders support dynamic content caching. Some of the popular ones are ASP.NET, BEA WebLogic, IBM WebSphere (IBM Corp,), Oracle 91AS. Server side caching can reduce server load and improve response time when the client stress is high. Client side solutions place caches very close to the client and thus the client and cache will be considered to be on the other point of the network. In case of edge side solutions, caches are placed in between the client and the server and most of the cases it would be close to the client. However, edge side caches are treated to be having their own point on the network.

As shown in Eq. (1) the total latency depends upon many factors. There are many solutions proposed for improving the

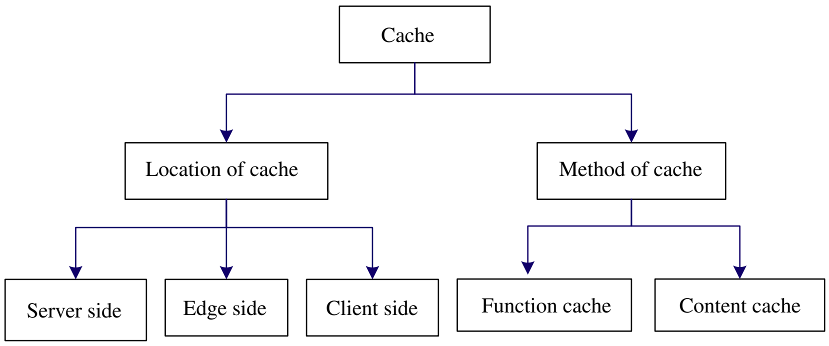

Fig. 3. A hierarchical classification of Web caching technology.

performance of DNS lookup L_{dns}. In this paper, however, we are not concentrating on that area. Latency introduced by the user machine is another area on which the research community can do little about. A higher speed cpu of the user machine will invoke a Web page faster than a lower speed cpu. Research community has concentrated on improving L_g, L_{nt} and L_{na} variables along with solutions to improve bandwidth availability, scalability and availability of the origin servers. In this section we concentrate on techniques which improve L_g, L_{nt} and L_{na}. Most of the server side caches focus on improving L_g. Most of edge caching methodologies tend to improve L_{nt} and most of the client side caching techniques improve L_{na}.

4.2. *Caching classification based on type of object being cached*

Based on the type of object being cached, we classify the caching solutions into two main types, viz., function caching and content caching. Both methods could be used for either server side proxies, client side proxies or proxy side caches.

In the content caching, the HTML page which the application generates is cached as different fragments/channels. These fragments/channels are maintained as separate objects in the edge server's cache and are dynamically assembled into Web pages using XML/JavaScript type languages by fetching only non-cacheable or expired fragments/channels from the origin server in response to user requests. The origin server supports this assembly and the exchange of information between origin server and proxy is XML type data. The representatives of this approach include Akamai Technologies Inc, CONCA Shi and Karamcheti (2001), and client side include (Rabinovich et al., 2003). All of them are built upon the edge side include (ESI) technology (Tsimelzon, 2000). In the function caching, the application itself is replicated and cached along with its associated objects so that the edge servers run applications instead of the origin server. The exchange of information between the origin server and the proxy is the state of the application itself. Examples of this method are vMatrix (Aggarwal and Rabinovich, 1998), IBM Websphere (IBM Corp.,), Active Cache (Cao et al., 1998), Gemini (Alzoubi et al., 2008), ACDN (Karbhari et al., 2002), Proxy+ (Yuan et al., 2004), and SEE (Mastoli et al., 2003). In the following sections we explain each technique in detail.

4.2.1. *Content caching*

The most popular solution lies in breaking the dynamic pages into fragments and then studying each individual fragments to understand their cacheability. Based on this concept many have proposed many different solutions. In all content caching scenarios a part or whole of the dynamic page will be shipped from the origin servers. The main job of the proxy is to assemble the pieces if pieces are shipped else manage and maintain the complete pages if full pages are shipped. In all these cases the work on the proxy is directly proportional to the size and number of pages that it handles. Hence CPU time is directly proportional to these variables. There are many content caching schemes available.

Among the server side content caches we have DUP algorithm: performance and cache hit has been improved dramatically by introducing DUP algorithm (Challenger et al., 1999, 2000) by keeping a data dependency information between cached objects and underlying data. Caches may contain entire HTML pages and also fragment of HTML pages. As soon as the system becomes aware of changes in the underlying data by way of triggers, graph traversal algorithms are applied to determine which cached objects are affected by the change. Based on invalidation, pages are pre-fetched instead of waiting for load on demand. This way the cache hit rate is as good as close to 100%. Since cache is close to the origin server, update of the cache is very quick and thus cache holds the most recent data. Prefetching does not involve much overhead on network bandwidth as the cache is very close to the server. Due to the proximity of the cache and also prefetching scalability of the system is very good. However, data storage source and cache are highly integrated and are implemented at the source end of the cache. To adopt the same at the network edge requires more complex integration with the origin server and implementation becomes vendor and application specific. This works for dynamic pages that are not personalized. Having a session build into the dynamic page generation is not addressed in this solution. DUP with its trigger tables could be complex when we have a combination of many fragments of HTML objects and also entire HTML pages which are associated with the database data. Every single database data could be associated with many fragments which in turn could make up complete HTML pages and thus linking of these HTML fragments with the underlying data could potentially become very complex with many data displayed in the same page or a fragment changing at different times. This could even lead to frequent invalidation and updating which could hurt performance.

Commercially available technologies: Content caching is the most widely applied caching methodology in the commercial world. The techniques that are used to implement content caching are unique to each technology. Some existing commercially available technologies are described as below.

Java Server pages technology and ASP.NET: Java Server Page and ASP.NET allow programmers to mark a part of a page as cacheable using tags, e.g., in JSP we can use a custom JSP tag "jc:cache" to cache a fragment. By using this tag, the application server will not generate that part of the fragment. Such fragments can be explicitly put into a user control which has its own cache parameters and can be included by pages or other user controls. However, the use of this would be mainly restricted to static portions of a dynamic page. For instance, any dynamic page would have a header, footer, and menu systems which would not change with time. However, since it is a part of a dynamic page, traditional servers will assemble them again and again whenever that page is invoked irrespective of the fact that portion of the page has not changed. To avoid such works on the server, these portions can be marked within the special tags. The advantage of using these technologies is that it is simple to implement and also it is very well accepted by the industry and implemented in modern applications. Though this is used mainly at the server end, the same can be extended to be workable at the edge servers as well provided the edge servers support these technologies. WebSphere suites can make use of the java tags and the edge suite can get the benefit by this implementation. However, this technique would be applicable only for non-personalized data.

Proxy+ is proposed by Microsoft research. The concept is to replicate the server side caching functionality to the proxies. Latest technologies as already explained above provide special tags for fragment caching which is used by the servers. This methodology uses the same tags to be used at the edges. This model consists of Web filter and a cache. Web filter is responsible for directing the caching of multiple versions of pages and fragments as well as composing pages. Cache is the storage of previously requested pages and fragments. When a HTTP request arrives at the proxy, filter computes the cache keys of the pages and its fragments. If all necessary items are valid in the cache the result will be returned immediately. Otherwise it attaches a list of keys identifying cached versions deemed relevant to the HTTP request header and forwards it to the server. Server uses "cache" tags to inform proxy that these are being cached. Proxy uses "X-cachedKeys" tag to inform about the already cached content so that redundant computation and transfer may be avoided. Since this is an extension of the existing protocol it is simple to use. No ill side effects are caused when a proxy+ aware application interacts with a normal proxy. The extra tags will be ignored by the normal proxy and no keys will be forwarded to the server which is just fine for the server functioning.

Disadvantage of these commercially implemented technologies is that each technology has its own set of tags for the usage. Filter and cache pair have to be uniquely designed to each technology. Scalability will be an issue with many different types of Web servers which are in the market today. Personalization is again not addressed here.

Edge side include technology: ESI is a simple markup language used to define Web page components for dynamic assembly and delivery of Web applications at the edge of the Internet. It enables dynamic assembly of Web pages composed of both cacheable and non-cacheable page fragments. It also contains a content invalidation specification for transparent content management across content delivery networks, application servers. For a dynamic Web page, the Web site has to pre-establish where ESI tag is used to define the content data and layout and their cache control attributes. ESI enables companies to develop Web applications once and choose at deployment time where the application should be assembled—on the content management system, the application server or the content delivery network. The ESI open standard specification is being co-authored by Akamai, ATG, BEA Systems, Circadence, Digital Island, IBM, Interwoven, Oracle, and Vignette.

Using ESI lets a content provider break a dynamic page into fragments with independent cacheability properties. These fragments are maintained as separate objects in the edge servers cache and are dynamically assembled into Web pages in response to users requests. A key advantage to using fragments is that when a fragment changes, the entire Web page does not have to be updated; only the fragment needs to be updated. This is an industry widely used caching technology which has the backing of big players like Akamai and Oracle, which uses a simple protocol based on markup language. This technology can be implemented on any part of the network, i.e., server end, edge server or client end of the network. However, it is more popularly used at network edge.

There are several techniques proposed by the research community which extrapolate ESI. Determining the fragment granularity level and how to fragment the pages are two key issues here. Shi et al. (2003) traced three news Web sites, two e-commerce Web sites and one entertainment site and discovered some characteristics of dynamic content: How to divide the Web page into object components without affecting the page freshness? When dividing page into components, the size of the component is not related to the freshness (this indicates the fundamental challenge on the fragment level assembly). It also shows that all of the objects either change on almost every access or change very infrequently, indicating a significant opportunity for content reuse both temporal and spatial ways, and the finer object granularity leads to more reusability. And the nature of freshness time distribution makes client-initiated caches consistency approach more appropriate than server initiated one. Brodie et al. (2005) also noticed another common behavior of dynamic content that one or more fragments move between the different positions across document.

There are a few proposals presented to automatically fragment the Web pages rather than manual approaches suggested earlier. Brodie et al. (2005) proposed two approaches which depend on keyword-based fragment detection. The technique uses predefined keywords to find these fragments and split them out of the core document. The DyCA, a dynamic content adapter, was implemented to take original dynamic Web content and convert it into fragment-based document. The other proposal, an augmentation to the ESI standards, allows splitting the information based on the position of each fragment in the template from the template data itself by using a mapping table. By using this keyword-based approach, a fragment enabled cache can have a finer grained level of identifying fragments independent of their location on the template, which enables it to take into account fragment behaviors such as fragment movement.

Ramaswamy et al. (2005) proposed a framework for automatic fragment detection, by including an augmented fragment tree with shingles encoding for modeling the dynamic Web pages. A so-called “shared fragment detection algorithm” can detect fragments shared across multiple documents. The work also suggests lifetime-personalization-based fragment algorithm to detect fragments which are most beneficial to caching based on the nature and the pattern of the changes occurring in dynamic Web pages.

While ESI seems promising for the Web site where cache can be application independent, it has issues with the dynamic page which requires server side application to support. If the fragment cache is determined only based on the request URL or other attribute like keyword, it may mistreat the validation.

The biggest drawback of the commercially implemented solutions is that personalization is not addressed. Hence all personalized page requests are forwarded to origin servers with no improvement in performance. Whenever authentication is required, Akamai Technologies Inc edge servers do not cache any data and the entire request is forwarded to the origin server. Also though ESI is widely accepted industry standard today, the user of ESI has to still redesign deployment of dynamic pages on his servers to make it ESI compatible for page dissemination.

Client side include technology: In this technology ESI fragments are extrapolated to be used at the client end with the client side include (Rabinovich et al., 2003) technology. Here ESI fragments are assembled at the client end. JavaScript/ ActiveX objects are used to run the applications on the browser which fetch the needed fragments from the origin server. This reduces the latency in the “last mile”, which is especially useful with dial-up clients who have slow connections. In this scheme, since the ESI fragments are cached at the client end, improvement in user perceived latency is the best among all the methods addressed so far. Since it uses the existing technologies like Javascript/Active X and applies it on exiting browsers hence deploying the idea is simple. However, if a client uses a different browser each time he accesses a Web page then this technique would be of little help. Also once the fragments are fetched at the client end it can be used by only one user unlike in edge side caching where a fetched fragment can be used by many clients who visit that edge cache.

AJAX-asynchronous JavaScript and XML: The most recent trend in the client site scripting is AJAX. In this, scripts which run on the client or on the remote server retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. The data could be either pre-fetched or the delta data alone could be retrieved which could be used to populate the next set of pages that the user browsers, which significantly reduces the user perceived latency. Castro et al. (2006) proposed client side state management for AJAX applications. In their solution, the application runs on the client browser and the data exchange between client and server happens after several interactions between client browser and the user. Domenech et al. (2006) proposed a pre-fetching algorithm which significantly reduces the user perceived latency. Furthermore, to enable rich content on mobile devices, Flashproxy (Moshchuk et al., 2008) applies a proxy-based approach which splices active content out of Web pages and replace it with an AJAX-based remote display component.

4.2.2. Function caching

There are many function caching solutions which are implemented in the commercial world. IBM WebSphere Edge Server V2.0 for Multiplatforms (edge server) distributes application processing to the edge of the network under centralized administrative and application control. Application offload shifts the burden of serving composed, personalized, dynamic content from the application server to edge servers placed at network edges by offloading back-end servers and peering links. Application is delayered into edgeable components. This also has transcoding capabilities and also enhanced cache for holding fragments. This product has both function and content caching capabilities built into the edge server. Personalization is addressed here. Using Tivoli SecureWay Policy Director, edge server itself can handle authorization. Load balancing of servers can also be achieved in this method. However, the system is tightly integrated with WebSphere software family and also is not an open protocol which others can use and implement.

On the above lines there is another solution proposed by Stanford University called VMatrix (Awadallah and Rosenblum, 2002). The basic idea behind VMatrix is that every dynamic page is a result of executing a program. Program which produces the dynamic page depends upon many variables which makes the page status to change. The concept here is to encapsulate all the variables and also the program into a virtual machine file. This file would be instantiated on any proxy which is hosting a virtual machine monitor. Virtual machine monitor virtualizes the real machine at the hardware layer and exports the virtual machine which mimics the real machine state. Each proxy is able to hold a users session and hence personalization and authorization are easy to implement on the proxy. This is an open protocol; however, commercial implementation of this protocol is still awaited.

University of Wisconsin proposes another function caching solution: active cache. Active cache scheme supports caching of dynamic contents at Web proxies. The scheme allows servers to supply cache applets to be attached with documents, and requires proxies to invoke cache applets upon cache hits to furnish the necessary processing without contacting the server. This function caching mechanism incurs less overhead in terms of network bandwidth and hence it incurs least communication overheads as the size of the function is usually independent from the object state and size. Hence even if the entire page has changed the size of the function transported is the same. It works best when the content of dynamic page changes drastically. However, the analogy is also true which is even if the change in the content of the page is very small then also the same size function needs to be shipped which might eat up on the efficiency of the system especially when the changes are very small. The disadvantage of this system is that every proxy should be capable of running the furnished applets.

There is another set of function caching which not only presents the information that the origin servers provide but also add value at the network edges. These include the transcoding services at edge servers as described by Bell Laboratories (Beck and Hofmann, 2001; Mastoli et al., 2003), collects per-user information and provides personalized services based on the user information at the edges. IETF's open pluggable edge service (OPES) framework (Barbir et al.) and content adaptation (Fox et al., 1998; Fu and Shi, 2001) encourage one to offload functions to edge services due to perceived advantages in terms of user perceived latency, load balance, availability of service to name a few. Content adaptation allows the system to inject additional functionalities along the data path between client and server. OPES proposes an environment to provide value added services to the end-users and/or content providers. Providing services at the edges enables incremental deployment and amortization of operating costs, thus benefiting the client and the provider both (Beck and Hofmann, 2001; Fox et al., 1998; Mastoli et al., 2003).

5. Application characteristics and caching solutions

The previous sections describe different solutions to improve different parameters of the latency (Eq. (1)). Caching is employed to improve performance in the areas as indicated in Fig. 4.

However, not all application may benefit from all these solutions. For example, the Web application which has high computational needs will not benefit from a content cache but a function cache would be a more ideal solution. The Web application which has sparse clients who use dial-up connection will benefit from client side caching than having a distributed network edge cache or any other cache which improves network bandwidth as what is needed for this application is a solution which reduces the latency in the last mile L_na, rather than reducing latency in any other leg viz., L_g, L_nt which contribute little to nothing to the overall user perceived latency compared to L_na which has higher influence on the overall latency. When deciding on which caching methodology to adopt, it would be more appropriate to study the application's dynamics and needs and identify the weak areas which has adverse influence on the overall performance.

With this background it is important to study the application's characteristics then choose an appropriate caching solution. The applications could be classified based on the size, nature, complexity of the architecture and the Web pages being hosted. The variable metrics to be considered while choosing a caching type can be broadly based on the frequency of update needed for the caches, application complexity, and application's computational needs, which are explained in detail below.

In this section, we classify and study the application needs and study the existing caching solutions which match the particular applications needs.

5.1. Based on application's computational needs

If an application has a high computational demand in which every single dynamic page is generated by performing intensive computations on the server side, then the L_g of the application

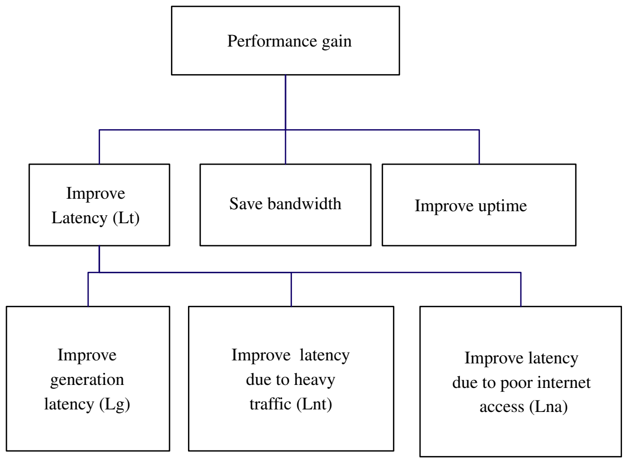
Performance gain
Improve Latency (Lt)
Save bandwidth
Improve uptime
Improve generation latency (Lg)
Improve latency due to heavy traffic (Lnt)
Improve latency due to poor internet access (Lna)

Fig. 4. The advantages of caching techniques.

would dominate over other variables in Eq. (1). Most of the dynamic page generations reflect the values in the database. There are, however, certain types of dynamic page generations, for example, online exams which have to compute the final score based on user inputs. Such pages which utilize heavy cpu times for dynamic page generation would benefit by using any of the function caching mechanisms. For all these applications, however, the proxy servers should be capable of handling the computation and should be able to communicate with the origin server to receive only function updates.

IBM WebSphere Edge Server V2.0 (IBM Corp,) for Multiplatforms (edge server) distributes application processing to the edge of the network. Origin server shifts the load on computation to the edge server and the edge server uses its cpu to generate the dynamic pages. VMatrix (Awadallah and Rosenblum, 2002) is another function caching solution which generates the page from the edge by executing a program on the virtual machine of every edge server. In both of these solutions each proxy is able to hold a users session and hence personalization and authorization are easy to implement on the proxy. Karve et al. (2006) evaluated a middleware clustering technology capable of allocating resources to Web applications through dynamic application instance placement. If a particular application has a computational demand for only a certain period of time, then the proposed solution is ideal. Application instances on a given set of server machines are placed to adjust the amount of resources available to applications in response to varying resource demands of application clusters. If only certain requests consume excessive computational power of the resource, Zhou and Yang (2006) proposed a termination scheme for threads which consume more resource so that threads which are computationally less intensive will be serviced faster thereby enhancing the QoS. When the heavy computation moves to the database queries, like, e.g., continuous aggregate queries used in online decision making applications, data mining applications, Gupta and Ramamritham (2007) proposed a content distribution network of dynamic data items. Just as various fragments of a dynamic Web page are served by one or more nodes of a content distribution network, their technique involves decomposing a client query into sub-queries and executing sub-queries on judiciously chosen data aggregators with their individual sub-query incoherency bounds. Their performance results show that cost-based query planning leads to queries being executed using less than one-third the number of messages required by existing schemes.

5.2. *Based on number of dynamic pages which change with the change in data*

Most of the dynamic pages reflect the data in the databases. However, data in a certain table could be susceptible to changes more frequently than others. If the design of the dynamic pages are such that frequently changing data are shown in certain pages and data which do not change so frequently are separated out, then we can decide on the type of caching solution for this mix of pages. For example, if we consider a stock exchange Web page, the user profile page which holds the user information does not change as often as the price of the stocks. If we divide the Web pages based on this category, then more efficient solutions can be proposed for frequently changing Web applications and different suitable solutions can be applied to less frequently changing Web applications.

5.2.1. *Few pages, about 0–20% of Web pages change with change in underlying data*

For applications which have low level of page updates, we can propose full page caching techniques like Challenger et al. (1999) and Candan et al. (2001). The commercial solutions include Internet security and acceleration (ISA) server from Microsoft, while hardware solutions are available from CacheFlow Gadde et al. (1997) and network appliance. These sit between the site and Internet, relieving the site from the work required to push responses through the site. Many other solutions are deployed in forward proxy mode, that is, in distributed caching architectures located at numerous points around the Internet, e.g., Gadde et al. (1997) and Aggarwal and Rabinovich (1998). There are also content delivery networking solutions, offered by vendors such as Akamai and Exodus which provide full page caching.

Since it is only a few pages that change due to change in the database data, techniques like ESI may not be warranted to be used though they might be more efficient in terms of reducing the user perceived latency. When we consider the overhead of maintaining individual fragments and the need to make the system ESI compatible, page level caching might work out better. This can be better explained in the following equation:

Total cost $(T_c) =$ bandwidth usage cost (B_c)+overhead on infrastructure to maintain a cache object (O_c)+cost in terms of time to generate the cached page (G_c).

In the case of page level caching G_c would be zero as no page needs to be generated by the cache; however, B_c might be high as the complete page is being transported. O_c is also on the lower side as the entire page is one single object. In case of ESI, B_c might be low as only fragments are being transported; however, G_c would be more than 0, O_c would be linearly proportional to the number of fragments which make up a page as every fragment is treated as an independent entity and the edge server needs that much more memory and resources to hold each fragment information.

5.2.2. *Many pages of the application (80–100%) change with change in underlying data*

Many of the news Web sites have pages which change very frequently. However, there is only a small portion of the dynamic page which gets affected due to change in the underlying data. For example, the head line news portion could change very often during the day while the local news portion does not change as often. In these cases page level caching will not work due to the huge overhead on bandwidth to carry the entire page where it is really not required as certain portions of the page would not have changed at all. In these cases, ESI, CSI, delta encoding-based solutions work the best in terms of efficiency for any edge-based proxy. DUP would work best for server side proxies.

As can be seen from the equation above, if the application has too many fragments which change with change in the underlying data, then ESI will tend to be very inefficient. The reason being every fragment introduces overheads for its maintenance in terms of maintaining its TTL, memory in the cache to save the information, etc. If a dynamic page has all of its fragments changing with any changes in database data, then maintaining the cache with too many fragments will fail heavily on the bandwidth between the server and the proxy in addition to extra load on the server to process each fragments. In all such cases a page level caching works best despite a few more updates that may be required due to not making it more granular.

5.2.3. *Moderate number of pages (20–80%) of the Web pages changes with change in underlying data*

For these applications a suitable solution from the above two categories can be chosen based on which end the application leans toward over a period of time.

5.3. *Based on the frequency of changes observed by the dynamic page*

The change frequency of dynamic pages becomes an important issue while deciding on the type of caching to adopt. We define high frequency change as a state when the page changes its content everyday several times a day. A page falls under medium frequency change when the page changes a few times in a month and low frequency change is applicable when the page changes few times a year.

5.3.1. *High frequency change*

If a page is changing every day many times in a single day, for instance a stock market page, then a caching mechanism which does not impose huge overheads on the network traffic would be needed. If there is any change in the data at the origin server, the origin server would have to maintain and manage the change and communicate the change very frequently to all its proxies. Before we decide on the solution, we first emphasize the need to study the network traffic and hence the usage of the bandwidth. The bandwidth usage can be broken into two parts. The first is the traffic created by the users to access the Web site to see the latest data. The second is the network traffic generated by the server to update and maintain all its replicas. The network traffic generated by the former is not under the control of the application. However, the traffic generated by the latter can be controlled based on the type of caching solution adopted. For example, Active Cache, vMatrix, PACE, and other function caching techniques generate less traffic in the latter as the traffic generated by shipping functions is far less than the traffic generated by shipping the whole pages. For more sophisticated, secure, multi-tier systems, IBM WebSphere and BEA WebLogic can be used. When the frequency of change is very high then any function caching methodology is most suited. Though the function shipping is small, if the dynamic page generated depends heavily on databases, then traffic due to maintaining consistency with distributed databases also comes into picture. However Ravi et al. (2005) have shown through their prototype PACE that the network traffic generated by sending just the database data is far less compared to the traffic generated by sending the complete pages which incorporate these data. This is especially true in case of dynamic pages which are generated to display emails. Studying a typical email page of Yahoo and Hotmail service providers as examples they have shown that a large amount of data is added to the actual required data on every single Web email page. Even if the email itself is 1 byte, usually a page of 18–40 kB is dynamically generated to display it. Hence caching 18–40 kB of data at the network edge and transporting the email from the origin server to the edge would significantly save bandwidth as shown by the prototype PACE.

Sivasubramanian et al. (2005) have shown that for applications which have high updates, replication of application along with data gives better performance along with wide area bandwidth savings. Hence for all applications which have high client access and dynamic pages of high frequency of changes, function caching would be the ideal solution. With high frequency change also requires keeping the content fresh. Li et al. (2003) proposed a freshness-driven adaptive dynamic content caching, which monitors the system status and adjusts caching policies to provide content freshness guarantees for a given set of user request rate, database update rate, and network latency parameters. The idea here is to use a DB cache which is a lightweight DBMS without the transaction management system and it may cache only a subset of the tables in the master database. All non-transactional requests are forwarded to DB cache which is present in all proxies. However, if the consistency can be relaxed as in the case of news Web sites where freshness of data can be compromised but high availability and performance is the key then fresh data, Gao et al. (2005) model of using the application specific semantics to design distributed objects using simple consistency models would work. They find that their object-based edge server system provides five times better response time over a traditional centralized cluster architecture and a factor of nine improvement over an edge service system that distributes code but retains a centralized database.

5.3.2. *Medium frequency change*

These dynamic pages change once in a week or a few times a month. In this case the traffic generated by the origin server to maintain consistency across caches due to change in the data is not very high. In every case there is strong need to study both the client traffic and the traffic generated by the server to maintain change consistency across proxies/caches. There are many advantages in using full page caches and ESI-based caching for these kind of applications. The main advantage is that using these systems, the scalability is very high, better availability, simple protocols and these will in turn translate to better pricing. Though we have not studied the price implication in this survey paper, using more sophisticated tools will generally translate to more price as well. Naaman et al. (2003) have shown that for low frequency changes delta encoding could be a good choice considering the fact that CDNs are not really required and a few hardware and software updates at the server and proxies can give good performance improvements. They recommend ESI for high frequency changes as delta encoding would suffer from high network traffic due to exchange of complete base files instead of fragments as in the case of ESI.

However, if the application needs security, authentication, authorization feature, multi-tiered, messaging, etc., then again IBM WebSphere, BEA WebLogic would be a good choice.

5.3.3. *Low frequency change*

For all applications where the frequency of change is not very high, these pages can be treated as static pages and all the caching methodologies used for static pages can be adopted here too. In fact if the frequency of change in the dynamic page is a few times a year, the best methodology would be to convert the page to a static page instead of a dynamic page and use the existing, conventional, proven static caching techniques which have been in the market for several years. However, if authentication and authorization are needed for these dynamic pages, it would be ideal to break the site into two halves, one half carrying all the static pages which do not need authorization/authentication, and the other half needing authorization requirements and providing two different solutions for each on the above lines.

5.4. *Based on locality of clients*

Client locality plays a very important role in deciding on the type of caching to choose. We decide on a different caching solution for each scenario as detailed below.

5.4.1. *Most of the clients are very close*

For the types of applications which have all the clients located in one area, there is no need to go in for distributed caches. In this scenario, network traffic from the server to the Internet would be directly proportional to the number of clients accessing the system. For every request, the server has to generate the dynamic page for each client. In this scenario any of the server side caching solutions would be the ideal choice. The server side cache could even pre-fetch the changed page from the server ahead of time. In DUP the authors have shown that using graph traversal algorithm, the dynamic page is made current even before the client hits the page. This has resulted in significant performance gain and also hit rate has been found to be 100%. Cache and the server are tightly integrated which helps in maintaining the consistency. Sivasubramanian et al. (2004) have shown that it is really not required to maintain consistency across all proxies when only a few proxies are invoked by the users. A significant performance improvement and bandwidth savings is noticed when replication is done on demand to selected proxies which are accessed the most. This function caching technique can be applied to applications with localized clients. Saroiu et al. (2002) have shown that a local proxy cache received almost the same cache hit compared to an Akamai server which suggests that for clients who are concentrated in a single area, Akamai would not really help in a significant way compared to deploying a local proxy cache.

5.4.2. *Clients are distributed all across the globe*

For applications which have clients who are distributed all over the globe but clients are sparse, then again a distributed cache which has the capacity to cater to tens of thousands of clients would not be of much use as the cache would always run below its capacity. It would be a good idea to also study the nature of the connection that the clients are using for connecting to the ISP providers. If the clients are using dial-up predominantly then in such cases any client side caching solutions would work the best. Solution proposed by CSI, caches the dynamic page on the client computer using ActiveX, JavaScript objects. Castro et al. (2006) proposed client side state management for AJAX applications where instead of downloading a succession of HTML pages, users download a single application and use that application for a long period of time. The application is not a set of HTML pages, but rather a single page that can possible modify its own presentation based on data exchanged with a server. Client contacts the server only for small data exchanges after a period of interaction with the user.

The network traffic from the client to the server would be only in terms of getting the fragments and not the whole dynamic page. This would improve the performance at the same time other cost intensive solutions of using distributed proxies is not warranted. There would be a computational overhead on the client to assemble the fragments; however, the benefits of reducing the latency due to the last mile of the network will outweigh the extra computation and perceivably the user perceived latency reduces overall, as shown by CSI. Even if the clients are having high speed connections then also CSI would be most ideal from the cost perspective as other high end caching solutions may not be cost effective.

5.4.3. *Many near clients and many dispersed clients*

When clients fall under all strata of classification viz., while many clients are widely distributed with many others close to the original server, then we have to provide solutions at all levels namely, server side, client side and proxy side. Based on the application complexity, security needs, and dynamic content mix, all the above solutions can be studied and then a comprehensive solution can be worked out. Alzoubi et al. (2008) proposed load aware IP anycast to solve routing the client request by selecting the CDN not only based on the closeness but also based on the network traffic to the CDN.

5.5. *Based on the dynamic page content mix*

Many dynamic page applications have a mix of personalized pages and non-personalized pages which make up the total dynamic pages. The ratio of personalized pages to non-personalized pages plays an important role in deciding which caching method to adopt. We explain the solutions based on this classification below.

Personalized pages need some degree of security. Session also needs to be maintained for such pages. Any of the ESI/CSI-based systems works best for non-personalized pages. However, ESI/CSI-based systems are not capable of maintaining a session and also authentication and authorization are still a research area in these systems. WebSphere, Weblogic, J2EE, .NET and any of the function caching methodologies like vMatrix, CONCA, active cache are best suited to host sessions and hence good for personalized pages. An application might have a need for a personalized page to just to greet the user or it may go in for more sophisticated functionality which might include the need to save credit card details of the user. Depending upon the applications need for security, a suitable secure system can be chosen as explained in the Section 5.7.

5.6. *Based on application complexity*

Applications can be broadly classified as two tier and multi-tier systems. Two tiered systems mainly generate dynamic pages to display the data in the database. The caching opportunity in case of a two tiered application is at two tiers viz., front-end (presentation tier) and back-end or the EIS tier.

Multi-tiered architectures are software models that extend the basic two-tiered client/server model to three or more tiers. In the basic two-tiered client/server model, the client requests services and the server provides services. There are software interfaces on the client and the server side that connect with one another to handle these interactions. From the caching perspective, in a multi-tiered model the scope for caching is in all tiers, for example, the user interface tier, middle tier objects and the database tier. If the system has a messaging tier, legacy tier, etc., then caching can be spread across to these tiers too. Padala et al. (2007) developed an adaptive resource control system that dynamically adjusts the shared resources to individual tiers in order to meet application level QoS. This solution is applicable to all tiers of an application.

Caching in a multi-tiered system (a.k.a multi-layered system) becomes very interesting and complicated due to the complexity of the system. Different caching techniques can be implemented at different applicable tiers of the system. Any multi-tiered application will have a front-end which is the presentation tier, consisting of dynamic pages which the clients invoke. The middle tier will be business objects which are basically the interfaces between the clients and the back-end systems. Java has EJB's which form the middle tier. The end tier is usually a database or a legacy system. To simplify the tier caching solutions, we divide the tiers into three main tiers viz., Presentation tier, middle tier and ESI tier. A multi-tier application can exploit caching solutions in all the three tiers where as, a two tier application can use the solutions given for EIS and presentation tiers. We explain the solutions in the three tiers as follows:

5.6.1. *Presentation tier caching*

All the caching techniques that we have explained so far cater to this tier. ESI, CSI, full page caching, etc., all work on concept of caching the pages that the user sees. All the other tiers of the application are like a black box while considering front-end caching methodologies as we are only concerned about how best to cache the generated pages to the user and present it to the user in the least possible time. Since we have explained this tier in detail elsewhere in this paper, we would like to cover the presentation tier caching that is implemented by the application server. The ones that we discuss here is specifically meant for the presentation tier of the application server. The techniques in the presentation tier caching is mainly based on caching HTML fragments. Many application servers provide this type of caching capability (e.g., WebSphere from IBM and WebLogic from BEA systems, which can mitigate delays due to presentation tier tasks. Solutions from vendors such as SpiderCache take the approach of caching dynamically generated pages within the site infrastructure. These solutions are similar to reverse proxy caches, except that they operate within the site infrastructure, and are typically implemented as plug-ins to the Web server.

5.6.2. Middle tier caching

In some scenarios, data-intensive Web-oriented applications can benefit from middle tier caching (i.e., caching data at locations separate from the central database where that information is stored and transactional operations are performed). The main goal of caching on the middle tier is to make data readily available to applications without repeated involvement from central database servers to provide the data. By keeping data cached at this tier, we can reduce the need to communicate back to the database tier for every request. This not only preserves the capacity at the database tier, but eliminates network traffic as well.

Sun has Java objects like EJB's which cater to these needs and Microsoft has COM/MTS or COM+ for the same. Users can user session EJB's to cache a user's session and reuse it during the generation of another dynamic page. With IBM Websphere and BEA Web logic which implement j2ee technology can use EJB's in a distributed environment thus still maintaining content and transactional consistency across proxies and thus enabling edge side proxies in applications.

5.6.3. Back-end (EIS tier) caching

When the number of client requests increases, this places a burden on the middle tier and the EIS tier. It is important to realize that while application servers excel at distributing application logic, they do very little to help load balance the application data. This is because application servers do not attempt to act like databases. The database is where the data resides, and it is the primary responsibility of the database to ensure transactional integrity.

While the application server can scale will all the above techniques of caching and replication to meet the demands of the user population, the database can act as a single entity and can become overburdened. In other words, each client request is ultimately translated into a database request. At some point, the middle tier is blocked and unable to handle additional load regardless of how many idle worker threads are sitting in the pool which are waiting to help the users in the presentation tier. The database's inability to scale with the middle tier renders many of the scalability features of the application server cluster ineffective. Chen and Iyengar (2003) have proposed a tiered system for serving differential content to improve performance. The idea here is to introduce service differentiation by routing the clients needing higher quality of service to certain databases carrying better service and routing other strata of clients to other strata of databases while still maintaining consistency across all strata of databases. Wei et al. (2008) have proposed service-oriented data denormalization to address scalability. They propose to break Web application data into multiple independent data services which simplifies each database workload and allows a much more efficient use of classical techniques.

Caching is most beneficial when the majority of data access operations are read-oriented. It is not viable to cache data on the middle tier if the data change with each user transaction. Lookup lists for items that change infrequently, such as product catalogs, benefit most from middle tier data caching. Caching static information is simple and is done intuitively. For example, instead of repeatedly asking a database to return a static table of valid ZIP codes, the application might read them once during initialization and cache the results to be returned to every request asking for this information. Dynamic caching, on the other hand, is a much more challenging problem to solve. It is important to understand how frequently the data to be cached changes. If the information is modified every three months, a static caching approach will probably work (provided the application is re-initialized at least once every three months). As the update interval becomes more frequent, more thought must be given on how to access the cache. For applications that are primarily read-oriented, caching will be effective. For applications that are primarily update-oriented, caching will be potentially of a limited value.

Many approaches have been proposed to accelerate dynamically generated content through ESI tier caching. These approaches are based on the idea of caching different components which make up EIS tier within the site architecture. Various types of data caching have been proposed, including caching result of database queries (Yagoub et al., 2000), caching WebViews (which are essentially query results wrapped in HTML or XML) (Labrinidis and Roussopoulos, 2003), and caching database tables, e.g., TimesTen Team (timesten team, 2002) and Altinel et al. (2003). Tolia and Satyanarayanan (2007) proposed caching technique using cryptographic hashing to detect similarities with previous results and show that this increases the throughput for data-intensive applications by ten fold. Database caching approaches can reduce some of the delays associated with query processing operations.

All the back-end caching approaches can improve user perceived latency. As they are tightly integrated with the Web servers, data consistency is at the highest level. Since they are cached at finer granularity, their reusability rate is very good and thus have high cache hit rates. Limitation of this approach is that since it is not scalable and hence cannot be distributed close to the client which results in adding network latency.

While back-end approaches guarantee the correctness of results and offer the advantages of fine-grained caching, they do not reduce bandwidth requirements. They just help in reducing the dynamic page generation latency introduced by the origin server. However, ke Larson et al. (2004) proposed MTCache for mid-tier database caching to improve system throughput and scalability by offloading part of the database read only queries to intermediate database servers that partially replicate data from the back-end server. MTCache is a prototype mid-tier database caching solution for SQL server.

5.7. Based on security requirements

Dynamic Web pages can be broadly classified based on the need for security. Many of the news sites, though dynamic have least security needs as every user browses the same set of pages and there is no need for having security built in. However when ever personalization is needed, some amount of security is enforced. When it comes to e-commerce sites where purchasing and payment features are added to the application, highest security would be enforced. We now study the existing solutions and apply them to each group.

5.7.1. *Low security needs*

When a Web application hosts dynamic pages which reflect data which are not specific to a specific user then there is no need to enforce security. For example, a news site is not specific to a particular person and thus every user will view the same data. At the same time these are dynamic pages since the news changes with time. For such needs any content-based caching systems explained above can be applied, in particular, full page cache, ESI and CSI are a good choice as it is widely accepted in the market and also simple to implement. Services can be hired easily as well.

5.7.2. *Medium security needs*

A Web application may have a need to know who is visiting a Web site-based on which, specific pages are generated. In this case a need for authentication comes into picture. Only authentic users would be able to view certain pages. For example, any news group Web site, other entertainment and fun groups like yahoo groups have a need for medium security. The reason for authentication for these sites would be the ability of an authentic user to view certain data. For example, in yahoo groups, an authentic group member would be able to post messages and view messages of other group members. The security is enforced so that the right person is authorized to use the group services. Such sites do not have any purchasing capabilities and hence the protocol used could be either HTTP and some times HTTPS. However, in many cases even HTTPS may not be really needed for such systems. In most of these site even if the Web site is hacked by a successful hacker, there would not be any information that could be used by the hacker to cause any damage to the end user. Hence high security is really not needed for such systems. For the creation of dynamic pages of such Web sites we would propose an upcoming system which handles authorization and authentication like proxy+, vMatrix and Conca could be a good choice. Mannan and van Oorschot (2008) proposed a scheme called IM-based privacy-enhanced content sharing (IMPECS) for personal Web content sharing. IMPECS enables a publishing user's personal data to be accessible only to his/her IM contacts. A user can put his/her personal Web page on any Web server he/she wants (vs. being restricted to a specific social networking Website), and maintain privacy of his/her content without requiring site-specific passwords.

5.7.3. *Highly secure system*

Online security is becoming increasingly important to companies that intend to build their business over the Internet, especially with the sudden boom of online marketplaces. Most e-commerce sites have very stringent security requirements. These systems would generally be an extension of the medium security systems. Not only do these Web sites provides personalized greetings to the user but also allow the user to use his/her credit cards to purchase services and items on their Web sites. These systems have a need to pass stringent security requirements to the caches and every cache would perform just like another origin server.

Programmers of such applications add digital signatures and data encryption to their e-commerce applications. Security softwares like digital signatures, online authentication and data encryption help secure transactions, carried out on popular online marketplaces and other e-commerce sites. These Web sites invariably use HTTPS as their protocol.

At the least a multi-tiered application would have: (1) Presentation services tier: responsible for gathering information from the user, sending the user information to the business services for processing, receiving the results of the business services processing, and presenting those results to the user. (2) Business services tier: responsible for receiving input from the presentation tier, interacting with the data services to perform the business operations that the application was designed to automate (e.g., income tax preparation and order processing), and sending the processed results to the presentation tier. (3) Data services tier: responsible for storage, retrieval, maintenance, and integrity of data. With multi-tiered applications, security needs to be enforced at every tier which drills down to every proxy. For example, in a J2EE application, security needs to be enforced at the Web server which generates the dynamic Web pages and thus responsible for the presentation tier, application server which holds the business objects which communicate with the Web server on one side and the EIS systems like database systems and legacy systems, messaging systems on the other end and hence responsible for the business services tier. Databases, EIS, legacy systems which make the final tier which actually hold the data. Hence every tier is important to be secure else the weakest security tier will be vulnerable for hacking which will render the high security implemented in other tiers useless. As an example if a user has entered a credit card information, then every tier carries this information at different points of time and hence any tier which is weak in its security would be vulnerable for hacking and hence security cannot be compromised at any tier and all tiers are to be treated equally with respect to security.

For all these kinds of systems, proven function/content caching solutions like IBM Web sphere, BEA Web logic,.NET suites of products would be ideal as they integrate very well with their edge server suites. However, the disadvantage with these systems is that they are not inter-operable as the protocols used by these systems are highly custom made for each suite and hence the user will be confined to one set of advantages or disadvantages that he is offered with each suite. Strong security implementation comes with decreased scalability. Manjhi et al. (2006) provided a solution for security-scalability tradeoff and propose a new scalability-conscious security design methodology that features: (a) compulsory encryption of highly sensitive data like credit card information and (b) encryption of data for which encryption does not impair scalability. Effective invalidation is an essential part of a effective cache. Manjhi et al. (2007) proposed invalidation clues which are especially applicable for secure data.

Though the clients can be broadly divided in the above fashion initially, some of the clients may tend to lean toward another type after a period of time. We intend to address such issues also while proposing a solution.

With the wide availability of content delivery networks, many e-commerce Web applications utilize edge cache servers to cache and deliver dynamic contents at locations much closer to users, avoiding network latency. By caching a large number of dynamic content pages in the edge cache servers, response time can be reduced, benefiting from higher cache hit rates.

The CONCA (Shi and Karamcheti, 2001) project addresses the creation of dynamic pages at the edges by dividing the content into channels and each channel is cached separately. CONCA proxy maintains a per-user information which is used for personalization. CONCA also carries out user authentication at the proxy along with addressing the issues of personalization with respect to the nomadic user.

6. Future directions

Now we are in a position to discuss the possible future directions of Web content generation and delivery techniques. We classify them into four categories, *trusted peer-to-peer Web caching and delivery*, *exploiting clients access patterns*, *considering the semantics of Web content*, and *correctness of caching*.

Trusted peer-to-peer Web caching and delivery: While Web application itself explores opportunities to accelerate content generation and distribution, the underlining Internet technologies never stopped evolution. In the recent years peer to peer (P2P) applications have become very popular with the wide acceptance of Kazaa, Napster Inc and Bittorrent (for audio/video file sharing) and now recently Skype (for voice services) to name a few. A recent study motes that P2P systems accounts for more than three fourth of the HTTP traffic (Saroiu et al., 2002). It further shows that a P2P cache in their environment achieved a significant savings in peak bandwidth compared to any other type of caching. P2P eliminates the stereotype that we have accepted so far in the industry viz., client and server. In a P2P system every node is a client and also a server. Hence the traditional caching methodologies for static and dynamic pages which are all based on the concept of client and server will no longer be applicable in P2P context. There are many research papers on P2P caching (Androutsellis-Theotokis and Spinellis, 2004); however, the research area for using P2P for dynamic content caching is still in a very initial stage. Globule proposed by Pierre and Steen is one of few efforts towards this direction, which is a collaborative content delivery network composed of end-user machines that operate in a P2P fashion (Pierre and van Steen, 2006). The issues concerned will be in terms of the peer being able to host a complicated application on itself along with trust evaluation of the peer which is one of the most important issues when it comes to P2P. How to trust a peer to host another users application? How to guarantee reachability of a peer? Liang and Shi (2005) address these issues in their study of a peer-to-peer Web server sharing application. Their solution is based on the concept of building a decentralized rating of each neighbor peer, considering the availability, trustworthiness, peer system capacity, the number of users served by the peer, and so on. Based on the trust value, a new interesting concept called "currency mechanism" is introduced which can be used by the individual peers to barter each others services. Kubiatowicz et al. (2000) proposed an architecture in which un trusted servers can also participate and the data is protected through redundancy and cryptographic techniques (Table 1).

Fortino and Mastroianni (2008) provided a brief survey of future trends in CDN's and conclude the future of CDN's lies in using P2P, Grid and Agent technologies as applied to a CDN network. The logic is in utilizing the strengths of each of these technologies, viz., dynamism and fault tolerance of a P2P system, robustness of a Grid technology, intelligent behavior and self-organization features of agent-based technologies. They provide research papers which discuss each of these separately as also all three applied to a CDN network as put forth in the UPGRADE-CDN workshop. They even discuss a P2P solution with a practical application domain of a health care system where they aggregate a patients information distributed across different peers using simple XML.

Exploiting client access patterns: The assumption about user access pattern drives the design for caching schema, replication replacement and consistence management. However, the real user access behavior may be different from the assumption. Moreover, if the reality is just the opposite of the assumption, the performance can be even worse than without caching implementation, as extra caching management cost occurs without any gain. Very few work addressed how to make the caching mechanism self manageable by learning and adapting to real time user access pattern change. It is especially important for the Web site with big user base where the user access pattern is not well predictable. On the other hand, modern Web applications usually categorize users into different roles which makes user behavior more predictable. In the role-based Web application design, each role is associated with one or more certain resources, where resource can be either a function, HTML page or any URL. These semantic knowledge can be leveraged on design for data replication and user request dispatch to edge service.

We feel detailed discussion on cloud computing is out of scope of this survey paper however, we would like to briefly put forth the concept of "pay-as-you-go" concept which is gaining recognition in the recent days. In this model, a particular client of a CDN provider would pay for the services of CDN based on the usage of bandwidth and data stored by the CDN. With a very short notice the capacity of the service can be enhanced. This is especially useful for services where the client access patterns are seasonal or sporadic with intermittent peak loads for which upfront investment in the infrastructure is not required instead the capacities could be increased for the time required and the services costs are directly proportional to the usage. CloundFront (Amazon Web Service,) is one such CDN which runs on a cloud.

Considering the semantic of content: Can content generated by one Web application be re-purposed for other independently developed ones? The movement of semantic Web technology tends to make the Web site a global content database, it suggests the opportunity for content reuse horizontally. Instead of traditional content reuse in consequent requests or sessions for one particular Web application, content can be discovered, shared and reused among applications, as semantic Web can express itself and be understood to other Web applications. In this sense, cache agents can collaborate globally, across the boundaries of domain and applications, and achieve the maximum of content reuse. As the sematic Web still in its infancy, there is little research on how to publish, discover the content for reusability purpose.

Correctness of caching: While considerable research efforts or research concentrated on content generation and delivery acceleration techniques, little study has been given to the correctness of these techniques themselves. When the techniques improve the

Table 1
Comparison of available solutions.

Types	Data change		Change frequency		Client locality		Web page mix		Application complexity		Security requirements		Caching type	
–	High	Low	High	Low	Local	Dist	Personalize	Regular	Multitier	Two tier	High	Low	Content	Func
Java Server page (J2EE)	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
ASP (.NET)	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
IBM Websphere	Yes	No	Yes	No	No	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes
ESI	Yes	No	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	No
AJAX	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No
DUP algorithm	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No
Proxy+	Yes	No	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes
CSI	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	yes	Yes	No
VMatrix	Yes	No	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes
Active cache	Yes	No	Yes	No	No	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes

performance of the Web applications, do they ensure the correctness of system? For some Web sites whose primary purpose is information publish, the correctness is equivalent to the freshness, which means user will get the dynamic content as fresh as he or she is supposed to have. When it comes to the Web application with business logic, the correctness means the business state is consistent regardless of what content generation and delivery technologies are used. For example, when the same use case scenario repeats, system should maintain the same business state at the end, with either lazy caching or active caching or without caching at all.

In a recent study by Souders (2008) on high performance Web sites, he noted that the design of the front-end plays a significant role in reducing the overall latency of the loading a page. He proposes several techniques which emphasize on rearranging the content of a page for high performance which includes putting style sheets in the top of the page, putting java scripts in the bottom, Gzip the components which are being accessed by the Web page, making fewer http requests by merging smaller pages into a bigger one, adding expires header, and avoid css expressions. He even proposed that every Web site should use a tool, YSlow which evaluates a given Web page and analyzed why a particular Web page is loading slowly. Just re-arranging the Web content of a page can significantly reduce the latency is the claim in this work. This is especially true for pages which have rich UI with many css, Java Scripts, and links accessing other objects. In such cases the latency due to accessing back-end is about 9% compared to 91% being the latency of simply rendering the rich UI which can be significantly reduced by applying rules as given in this paper which includes re-arranging the data as one of the important rule.

7. Summary

Responsiveness is the most important success factor for Web application. Researchers and practitioners haven been trying all means to improve it where caching in the content generation and delivery is the key technology. Considerable solutions are developed for all aspects of applications, from the location of server, client and proxy sides, tiers of architectural layers, to processes of page fragmenting and assembly. A particular caching methodology could be very effective for a given application and less effective for others. Or it improves the performance on one aspect but has no effect on others. The amount of research work devoted to studying the different solutions available in the market but in terms of evaluating their effectiveness to a specific application as a whole it is not yet sufficient. If breaking up an application into modules, an effective dynamic caching solution for an application can only be studied based on how effective the solution is across all modules. The ideal approach would be to see the effectiveness of each individual solution against each module of the application and then see the overall effectiveness from the context of the complete application. This drives the need for more research efforts that do not assume the modules and solutions to be independent of each other.

References

Amazon Web Service ⟨http://aws.amazon.com/⟩.

Aggarwal A, Rabinovich M. Performance of dynamic replication schemes for an internet hosting service, 1998 ⟨citeseer.ist.psu.edu/aggarwal98performance.html⟩.

Akamai Technologies Inc. ⟨http://www.akamai.com/⟩.

Altinel M, Bornhoevd C, Krishnamurthy S, Mohan C, Pirahesh H, Reinwald B. Cache tables: paving the way for an adaptive database cache. In: Proceedings of the 29th international conference on very large data bases, September 2003 ⟨http://www.vldb.org/conf/2003/papers/S22P01.pdf⟩.

Alzoubi HA, Lee S, Rabinovich M, Spatscheck O, der Merwe JV. Anycast CDNs revisited. In: WWW '08: proceedings of the 17th international conference on World Wide Web. New York: ACM; 2008. p. 277–86.

Amiri K, Park S, Tewari R, Padmanabhan S. Dbproxy: a dynamic data cache for Web applications. In: ICDE04: Proceedings of the 19th international conference on data engineering, 2003. p. 821–31.

Amza C, Chanda A, Cecchet E, Cox A, Elnikety S, Gil R, Marguerite J, Rajamani K, Zwaenepoel W. Specification and implementation of dynamic Web site benchmarks, 2002 ⟨citeseer.ist.psu.edu/amza02specification.html⟩.

Amza C, Soundararajan G, Cecchet E. Transparent caching with strong consistency in dynamic content Web sites. In: ICS '05: Proceedings of the 19th annual international conference on supercomputing. New York: ACM Press; 2005. p. 264–73.

Androutsellis-Theotokis S, Spinellis D. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys 2004;36(4):335–71.

Awadallah A, Rosenblum M. The vMatrix: a network of virtual machine monitors for dynamic content distribution. In: Proceedings of the 7th international workshop on Web caching and content distribution (WCW'02), August 2002.

Barbir A, Penno R, Chen R, Hofmann M, Orman H. An Architecture for Open Pluggable Edge Services (OPES), Internet RFC3835 ⟨http://www.ietf.org/rfc/rfc3835.txt⟩.

BitTorrent—Peer to peer file sharing application ⟨http://www.bittorrent.com⟩.

Beck A, Hofmann M. Enabling the internet to deliver content-oriented services. In: Proceedings of the 6th international workshop on Web caching and content distribution (WCW'01), June 2001 ⟨http://www.cs.bu.edu/techreports/2001-017-wcw01-proceedings/107_beck.pdf⟩.

Bent L, Rabinovich GM, Voelker Z. Xiao, Characterization of a large Web site population with implications for content delivery. In: WWW '04: Proceedings of the 13th international conference on World Wide Web. New York: ACM Press; 2004. p. 522–33.

Breslau L, Cao P, Fan L, Phillips G, Shenker S. Web caching and zipf-like distributions: evidence and implications. In: Proceedings of IEEE conference on computer communications (INFOCOM'99), 1999. p. 126–34 ⟨http://www.research.att.com/~breslau/papers/zipf.ps.gz⟩.

Brodie D, Gupta A, Shi W. Accelerating dynamic Web content delivery using keyword-based fragment detection. Journal of Web Engineering 2005;4(1): 79–99.

Candan KS, Li W-S, Qiong W-P, Agrawal HD. Enabling dynamic content caching for database-driven Web sites. In: Proceedings of the 2001 ACM SIGMOD international conference on management of data, 2001 ⟨http://portal.acm.org/citation.cfm?id=375736⟩.

Cannataro M, Cuzzocrea A, Mastroianni C, Ortale R, Pugliese A. Modeling adaptive hypermedia with an object-oriented approach and xml. In: 2nd international workshop on Web dynamics, 2002.

Cao P, Zhang J, Beach K. Active cache: caching dynamic contents on the Web. In: Proceedings of IFIP international conference on distributed systems platforms and open distributed processing, 1998. p. 373–88 ⟨http://www.cs.wisc.edu/~cao/papers/active-cache.ps⟩.

Castro P, Giraud F, Konuru R, Ponzo J, White J. Before-commit client state management services for ajax applications. Hotweb 2006;0:1–12.

Challenger J, Iyengar A, Dantzig P. A scalable system for consistently caching dynamic Web data. In: Proceedings of IEEE conference on computer communications (INFOCOM'99), March 1999.

Challenger J, Iyengar A, Witting K, Ferstat C, Reed P. A publishing system for efficiently creating dynamic Web content. In: Proceedings of IEEE conference on computer communications (INFOCOM'00), March 2000.

Chen H, Iyengar A. A tiered system for serving differentiated content. World Wide Web: Internet and Web information systems, vol. 6(4), December 2003 ⟨http://www.research.ibm.com/people/i/iyengar/wwwj03.pdf⟩.

Davis A, Parikh J, Weihl WE. Edgecomputing: extending enterprise applications to the edge of the internet. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on alternate track papers and posters. New York: ACM Press; 2004. p. 180–87.

Domenech J, Gil J, Sahuquillo J, Pont A. Ddg: an efficient prefetching algorithm for current Web generation. Hotweb 2006;0:1–12.

Douglis F, Haro A, Rabinovich M. HPP:HTML macro-pre-processing to support dynamic document caching, In: Proceedings of the 1st USENIX symposium on Internet technologies and systems (USITS'97), December 1997. p. 83–94 ⟨http://www.douglis.org/fred/work/papers/hpp.pdf⟩.

Feldmann A, Caceres R, Douglis F, Glass G, Rabinovich M. Performance of Web proxy caching in heterogeneous bandwidth environments. In: Proceedings of IEEE conference on computer communications (INFOCOM'99), March 1999. p. 107–16. ⟨http://www.douglis.org/fred/work/papers/hetproxcache.pdf⟩.

Fortino G, Mastroianni C. Special section: enhancing content networks with P2P, grid and agent technologies. Future Gener. Comp. Syst. 2008;24(3): 177–9.

Fox A, Gribble S, Chawathe Y, Brewer EA. Adapting to network and client variation using infrastructural proxies: lessons and prespectives, IEEE Personal Communication 1998 ⟨http://www.cs.washington.edu/homes/gribble/papers/adapt.ps.zip⟩.

Fu X, Shi W, Akkerman A, Karamcheti V. CANS: composable, adaptive network services infrastructure In: Proceedings of the 3rd USENIX symposium on Internet technologies and systems (USITS'01), March 2001. p. 135–46.

Google Inc. ⟨http://www.google.com⟩.

Gadde S, Rabinovich M, Chase J. Reduce, reuse, recycle: an approach to building large internet caches. In: Workshop on hot topics in operating systems, 1997

⟨http://citeseer.ist.psu.edu/cache/papers/cs/8247/http:zSzzSzwww.cs.duke.e-duzSz~chasezSzcps216zSzcrisp.pdf/gadde97reduce.pdf⟩.

Gao L, Dahlin M, Nayate A, Zheng J, Iyengar A. Improving availability and performance with application-specific data replication. IEEE Trans. Knowl. Data Eng. 2005;17(1):2005.

Groothuyse T, Sivasubramanian S, Pierre G. Globetp: template-based database replication for scalable Web applications. In: WWW '07: Proceedings of the 16th international conference on World Wide Web. New York: ACM; 2007. p. 301–10.

Gupta R, Ramamritham K. Optimized query planning of continuous aggregation queries in dynamic data dissemination networks. In: WWW '07: Proceedings of the 16th international conference on World Wide Web. New York: ACM; 2007. p. 321–30.

Hibernate—Relational persistence for Java and .NET ⟨http://www.hibernate.org⟩.

IBM Corp. Websphere platform ⟨http://www.ibm.com/websphere⟩.

Inderjeet Singh MJ, Stearns B. Designing enterprise applications with the J2EE platform, 2nd ed. Reading, MA: Addison-Wesley; 2002.

Inria EC. C-jdbc: a middleware framework for database clustering ⟨citeseer.ist.psu.edu/682076.html⟩.

Kazaa—Distributed peer to peer file sharing service ⟨http://www.kazaa.com⟩.

Karbhari P, Rabinovich M, Xiao Z, Douglis F. Acdn: a content delivery network for applications. In: SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on management of data. New York: ACM Press; 2002. p. 619–19.

Karve A, Kimbrel T, Pacifici G, Spreitzer M, Steinder M, Sviridenko M, Tantawi A. Dynamic placement for clustered Web applications. In: WWW '06: Proceedings of the 15th international conference on World Wide Web. New York: ACM; 2006. p. 595–604.

Kubiatowicz J, Bindel D, Chen Y, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. Oceanstore: an architecture for global-scale persistent storage. In: Proceedings of ACM ASPLOS, November 2000.

Labrinidis A, Roussopoulos N. Balancing performance and data freshness in Web database servers. In: Proceedings of the 2004 international workshop on information quality in information systems, September 2003 ⟨http://www.db.cs.cmu.edu/Seminar/Summer2003/aug27⟩.

ke Larson P, Goldstein J, Zhou J. Transparent mid-tier database caching in sql server. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data, June 2004 ⟨http://portal.acm.org/citation.cfm?id=872848⟩.

Li W-S, Hsiung W-P, Po O, Hino K, Candan KS, Agrawal D. Challenges and practices in deploying Web acceleration solutions for distributed enterprise systems. In: WWW '04: Proceedings of the 13th international conference on World Wide Web. New York: ACM Press; 2004. p. 297–308.

Li W-S, Po O, Hsiung W-P, Candan KS, Agrawal D. Engineering and hosting adaptive freshness-sensitive Web applications on data centers. In: WWW '03: Proceedings of the 12th international conference on World Wide Web. New York: ACM Press; 2003. p. 587–98.

Liang Z, Shi W. Enforcing cooperative resource sharing in untrusted peer-to-peer environment. ACM J Mobile Netw Appl (MONET) 2005;10(6):771–83.

Liu F, Zhao Y, Wang W, Makaroff D. Database server workload characterization in an e-commerce environment. In: MASCOTS 2004: Proceedings of the IEEE computer society's 12th annual international symposium on modeling, analysis, and simulation of computer and telecommunications systems. Silver Spring, MD: IEEE Computer Society, 2004. p. 475–83.

Luo Q, Krishnamurthy S, Mohan C, Pirahesh H, Woo H, Lindsay BG, Naughton JF. Middle-tier database caching for e-business. In: SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on management of data. New York: ACM Press; 2002. p. 600–11.

Mahdavi M, Shepherd J. Enabling dynamic content caching in Web portals. In: RIDE '04: Proceedings of the 14th international workshop on research issues on data engineering: Web services for E-commerce and E-government applications (RIDE'04). Silver Spring, MD: IEEE Computer Society; 2004. p. 129–36.

Manjhi A, Ailamaki A, Maggs BM, Mowry TC, Olston C, Tomasic A. Simultaneous scalability and security for data-intensive Web applications. In: Proceedings of SIGMOD, 2006. p. 241–52.

Manjhi A, Gibbons PB, Ailamaki A, Garrod C, Maggs BM, Mowry TC, Olston C, Tomasic A, Yu H. Invalidation clues for database scalability services. ICDE 2007;0:316–25.

Mannan M, van Oorschot PC. Privacy-enhanced sharing of personal content on the Web. In: WWW '08: Proceeding of the 17th international conference on World Wide Web. New York: ACM; 2008. p. 487–96.

Mastoli V, Desai V, Shi W. SEE: a service execution environment for edge services. In: Proceedings of the 3rd IEEE workshop on internet applications (WIAPP'03), June 2003.

Mikhailov M, Wills CE. Change and relationship-driven content caching, distribution and assembly. Technical Report WPI-CS-TR-01-03, Computer Science Department, WPI, March 2001 ⟨http://www.cs.wpi.edu/~cew/papers/tr01-03.pdf⟩.

Mohan C. Caching technologies for Web applications. In: VLDB '01: Proceedings of the 27th international conference on very large data bases. Los Altos, CA: Morgan Kaufmann; 2001. p. 726.

Moshchuk A, Gribble SD, Levy HM. Flashproxy: transparently enabling rich Web content via remote execution. In: MobiSys '08: Proceeding of the 6th international conference on mobile systems, applications, and services. New York: ACM; 2008. p. 81–93.

Myers A, Chuang J, Hengartner U, Xie Y, Zhang W, Zhang H. A secure and publisher-centric Web caching infrastructure. In: Proceedings of IEEE conference on computer communications (INFOCOM'01), April 2001.

Naaman M, Garcia-Molina H, Paepcke A. Evaluation of delivery techniques for dynamic Web content, Poster. In: Proceedings of 12th international World Wide Web conference, May 2003 ⟨http://www2003.org/cdrom/papers/poster/p268/Poster.htm⟩.

Napster Inc. ⟨http://www.napster.com⟩.

ORacle Corp. Oracle application server ⟨http://www.oracle.com/appserver/index.html⟩.

Padala P, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K. Adaptive control of virtualized resources in utility computing environments. SIGOPS Oper Syst Rev 2007;41(3):289–302.

Pathan M, Buyya R, Vakali A. CDNs: state of the art, insights, and imperatives. Content delivery networks. Germany: Springer; 2008.

Pierre G, van Steen M. Globule: a collaborative content delivery network. IEEE Commun Magazine 2006;44(8):127–33.

Plattner C, Alonso G. Ganymed: scalable replication for transactional Web applications. In: Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware. New York: Springer; 2004. p. 155–74. ⟨http://portal.acm.org/citation.cfm?id=1045658.1045671⟩.

Rabinovich M, Spatscheck O. Web caching and replication. Reading, MA: Addison-Wesley; 2002.

Rabinovich M, Xiao Z, Douglis F, Kamanek C. Moving edge side includes to the real edge—the clients. In: Proceedings of the 4th USENIX symposium on Internet technologies and systems (USITS'03), March 2003.

Ramaswamy L, Iyengar A, Liu L, Douglis F. Automatic fragment detection in dynamic Web pages and its impact on caching. IEEE Trans Knowl Data Eng 2005;17(6):859–74.

Ravi J, Shi W, Xu C. Personalized email management at network edges. IEEE Internet Comput. 2005;9(2).

Ricadela, A., Microsoft goes alive ⟨http://www.informationweek.com⟩, November 2005.

Saroiu S, Gummadi KP, Dunn RJ, Gribble SD, Levy HM. An analysis of internet content delivery systems. In: Proceedings of the 5th USENIX symposium on operating systems, design and implementation, December 2002.

Shi W, Collins E, Karamcheti V. Modeling object characteristics of dynamic Web content. J Parallel Distrib Comput 2003;63(10):963–80.

Shi W, Karamcheti V. CONCA: an architecture for consistent nomadic content access. In: Workshop on cache, coherence, and consistency (WC3'01), June 2001.

Sivasubramanian S, Pierre G, van Steen M. Replicating Web applications on-demand. In: Proceedings of the IEEE international conference on services computing, September 2004.

Sivasubramanian S, Pierre G, van Steen M. Autonomic data placement strategies for update-intensive Web applications. In: Proceedings of the international workshop on advanced architectures and algorithms for Internet delivery and applications, June 2005 ⟨http://www.globule.org/publi/ADPSUIWA_aaaidea2005.html⟩.

Sivasubramanian S, Pierre G, vanSteen M, Alonso G. Analysis of caching and replication strategies for Web applications. IEEE Internet Comput 2007;11(1):60–6.

Skype—Peer to peer voice service ⟨http://www.skype.com⟩.

Souders S. High performance Web sites. Queue 2008;6(6):30–7.

Soundararajan G, Amza C. Using semantic information to improve transparent query caching for dynamic content Web sites. DEEC 2005:132–8.

Timesten team. Middle-tier database caching for e-business. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, September 2002 ⟨http://portal.acm.org/citation.cfm?doid=564691.564761⟩.

Tolia N, Satyanarayanan M. Consistency-preserving caching of dynamic database content. In: WWW '07: Proceedings of the 16th international conference on World Wide Web. New York: ACM; 2007. p. 311–20.

Tsimelzon M. ESI language specification, 1.0 2000 ⟨http://www.esi.org⟩.

Wei Z, Dejun J, Pierre G, Chi C-H, van Steen M, Service-oriented data denormalization for scalable Web applications. In: Proceedings of the 17th international World Wide Web conference, April 2008 ⟨http://www.globule.org/publi/SODDSWA_www2008.html⟩.

Wolman A, Voelker GM, Sharma N, Cardwell N, Karlin A, Levy HM. On the scale and performance of cooperative Web proxy caching. In: Proceedings of 17th ACM symposium on operating systems principles (SOSP), December 1999. p. 16–31.

Yahoo Inc. ⟨http://www.yahoo.com⟩.

Yagoub K, Florescu D, Issarny V, Valduriez P. Caching strategies for data-intensive Web sites. In: Proceedings of the 26th international conference on very large data bases, May 2000 ⟨http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=672020⟩.

Yuan C, Chen Y, Zhang Z. Evaluation of edge caching/offloading for dynamic content delivery. IEEE Trans Knowl Data Eng 2004;16(11):1411–23.

Zhou J, Yang T. Selective early request termination for busy internet services. In: WWW '06: Proceedings of the 15th international conference on World Wide Web. New York: ACM; 2006. p. 605–14.

Zhu H, Yang T. Class-based cache management for dynamic Web content. In: Proceedings of IEEE conference on computer communications (INFOCOM'01), April 2001 ⟨http://www.cs.ucsb.edu/projects/swala/cache2001.ps⟩.

Zhu Z, Mao Y, Shi W. Workload characterization of uncacheable http content. In: Web engineering: 4th international conference, ICWE. Berlin: Springer GmbH; 2004. p. 319–95.