Chapter 6

Integration of Database and Internet Technologies for Scalable End-to-end E-commerce Systems

K. Selçuk Candan
Arizona State University

Wen-Syan Li
C&C Research Laboratories, NEC USA, Inc

ABSTRACT

The content of many web sites change frequently. Especially in most e-commerce sites, web content is created on request, based on the current state of business processes represented in application servers and databases. In fact, currently 25% of all web content consist of such dynamically generated pages, and this ratio is likely to be higher in e-commerce sites. Web site performance, including system up-time and user response time, is a key differentiation point among companies that are eager to reach, attract, and keep customers. Slowdowns can be devastating for these sites as shown by recent studies. Therefore, most commercial content-providers pay premium prices for services, such as content delivery networks (CDNs), that promise high scalability, reduced network delays, and lower risk of failure. Unfortunately, for e-commerce sites, whose main source of content is dynamically generated on demand, most existing static-content based services are not applicable. In fact, dynamically generated content posses many new challenges for the design of end-to-end (client-to-server-to-client) e-commerce systems. In this chapter, we discuss these challenges and provide solutions for integrating Internet services, business logic, and database technologies, and for improving end-to-end scalability of e-commerce systems.

INTRODUCTION

The content of many web sites change frequently: (1) entire sites can be updated during a company restructuring or during new product releases; (2) new pages can be created or existing pages can be removed as incremental changes in the business data or logic, such as inventory changes, occur; (3) media contents of the pages can be changed the HTML contents are left intact, for instance when advertisements are updated; and (4) (sub)content of pages can be dynamically updated, for instance when product prices change. Some of these changes are administered manually by webmasters, but most are initiated automatically by the changes in the underlying data or application logic. Especially in most e-commerce sites, web content is created on-request, based on the current state of business processes represented in application servers and databases. This requires close collaboration between various software modules, such as web servers, application servers, and database servers (Figure 1), as well as Internet entities, such as proxy servers.

Web site performance is a key differentiation point among companies and e-commerce sites eager to reach, attract, and keep customers. This performance is measured using various

Figure 1: Database driven dynamic content delivery versus static content delivery

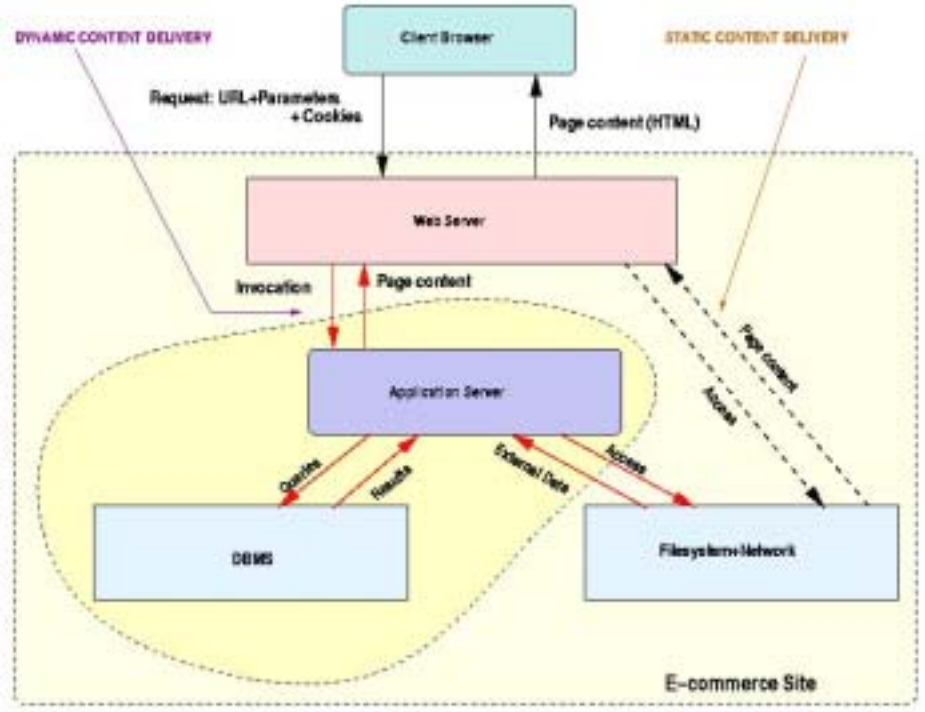


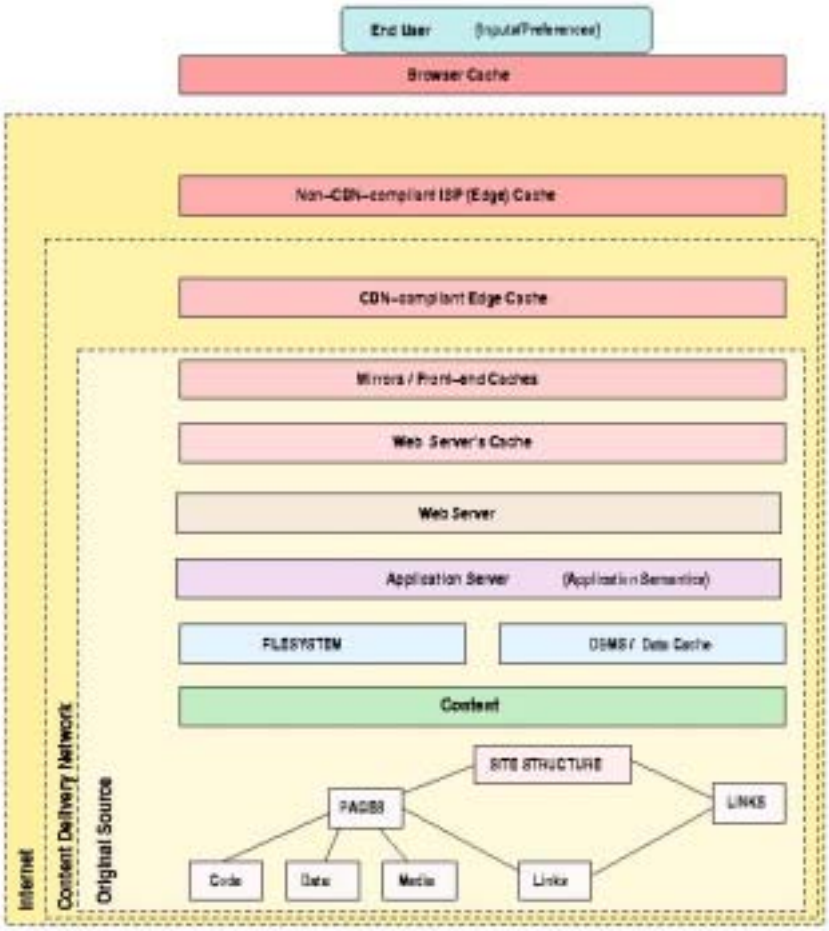
Table 1: Relationship between the time required to download a page and the user abandonment rate

Download time	Abandonment rate
< 7 seconds	7%
8 seconds	30%
12 seconds	70%

metrics, including system up-time, average response time, and the maximum number of simultaneous users. Low performance, such as slowdowns, can be devastating for content providers, as shown by recent studies (Zona Research, 2001), which indicate that even with response times of 12 seconds, web sites find 70% abandonment rates (Table 1).

As a result, most commercial web sites pay premium prices for solutions that help them reduce their response times as well as risks of failure when faced with high access rates. Most high volume sites typically deploy a large number of servers and employ hardware- or software-based load balancing components to reduce the response time of their servers. Although they guarantee better protection against surges in demand, such localized solutions can not help reduce the delay introduced in the network during the transmission of the content to end-users. In order to alleviate this problem, content providers also replicate or mirror their content at *edge caches;* i.e., caches that are close to end users. If a page can be placed in a cache closer to end-users, when a user requests the page, it can be delivered promptly from the cache without additional communication with the Web server, reducing the response time. This approach also reduces the load on the original source as some of the requests can be processed without accessing the source.

Figure 2: Components of a database driven Web content delivery system



This gives rise to a multi-level content delivery structure, which consists of (1) one or more local servers and reverse proxies, which use load distribution techniques to achieve scalability, (2) content delivery networks (CDNs) paid by the e-commerce site, which deploy network wide caches that are closer to end-users, (3) caches and proxy servers that are employed by Internet service providers (ISPs), which are aimed at reducing the bandwidth utilization of the ISPs, and (4) browser caches which store content frequently used by the user of the browsers. This structure is shown in Figure 2. Note that, although different caches in this structure are deployed by different commercial entities, such as CDNs and ISPs, with different goals, the performance of an e-commerce site depends on the successful coordination between various components involved in this loosely-coupled structure.

A static page, i.e., a page which has not been generated specifically to address a user request and which is not likely to change unpredictably in the future, can easily be replicated and/or placed in caches for future use. Consequently, for such content, the hierarchy shown in Figure 2 works reasonably well. For instance caching works, because since content is assumed to be constant for a predictable period of time, it can be stored in the caches and proxies distributed in the network without risking staleness of accesses. In addition, CDNs provide considerable savings on network delays, because static content is media rich.

For dynamically generated pages, however, such assumptions do not always hold (Table 2). One major characteristic of this type of content is that they are usually text oriented, and therefore small (4k). Consequently, the delay observed by the end-users are less sensitive to the network bottlenecks compared with large media objects.

In contrast, the performance of dynamic content-based systems is extremely sensitive to the load variations in the back-end servers. The number of concurrent connections a web server can simultaneously maintain is limited and that new requests have to wait at a queue until old ones are served. Consequently, system response time is a function of the maximum number of concurrent connections and the data access/processing time at the back-end systems. Unfortunately, the underlying database and application servers are generally not as scalable as the web servers: they can support fewer concurrent requests and they require longer processing times. Consequently, they become bottlenecks before the web servers and the network; hence, reducing the load of application and database servers is essential.

Furthermore, since the application servers, databases, web servers, and caches are independent components, it is not trivial to reflect the changes in data (stored in the databases) to the cached web pages that depend on this data. Since, most e-commerce applications are sensitive to the freshness of the information provided to the clients, most application servers have to specify dynamically generated content as non- cacheable or make them expire immediately. Hence, caches can not be useful for dynamically generated content.

Table 2: Dynamic content versus static content

common...	Static Content	Dynamic Content
Format	Text, Images, Video, Audio	Mostly text
Storage	File system	Databases (data) and application servers (business logic)
Source of Delay	Network delay	Data access and application processing delay
Scalability Bottleneck	Web server	Database and application server

Consequently, repeated requests to dynamically generated Web pages with the same content result in repeated computation in the backend systems (application and database servers).

In fact dynamically generated content posses many new challenges to the efficient delivery of content. In this chapter, we discuss these challenges and provide an overview of the content delivery solutions developed for accelerating e-commerce systems.

OVERVIEW OF CONTENT DELIVERY ARCHITECTURES

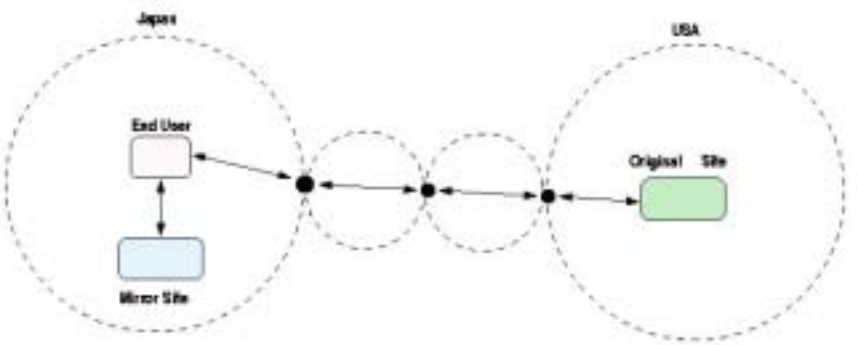
Slowdowns observed by major web sites, especially during their peak access times, demonstrate the difficulty companies face trying to handle large demand volumes. For e-commerce sites, such slowdowns mean that potential customers are turned away from the electronic stores even before they have a chance to enter and see the merchandise. Therefore, improving the scalability of web sites is essential to companies and e-commerce sites eager to reach, attract, and keep customers.

Many e-commerce sites observe non-uniform request distributions; i.e., although most of the time the request load they have is manageable, in certain occasions (for example during Christmas for e-shops and during breaking news for news services) the load they receive surges to very high volumes. Consequently, for most companies, investing in local infrastructure that can handle peak demand volumes, while sitting idle most other times, is not economically meaningful. These companies usually opt for *server farm-* or *edge-based* commercial services to improve scalability.

Server Farms vs. Edge Services

Server *farms,* provided by various companies including Digital Island (Digital Island, 2001), Exodus (Exodus Communications, 2001), MirrorImage (Mirror Image Internet, Inc., 2001), are one possible external scalability solution. A server *farm,* in essence, is a power-house that consists of hundreds of colocated servers. Content providers *publish,* or upload, the content into the server farm. It is then the responsibility of the server farm to allocate enough resources to ensure a quality of service to its customers. Note that, by their nature, server farms are expensive to create. Therefore, companies that provide such services tend

Figure 3: Network delays observed by the end-users



to establish a few server farm sites, usually at sites closer to where most of their content providers are. These sites are then linked with high bandwidth leased land or satellite connections to enable distribution of data within the server farm network. Some of the server farm companies also provide *hosting* services, where they host the entire site of their customers, relieving them from the need of maintaining a copy of the web site locally.

Although the server farm approach provide protection against demand surges, by leveraging the differences between the demand characteristics of different content providers, since the farms can not permeate deep into the network, it can not reduce the network distance between the end-users and the data sources (farms). This, however, may contribute to the overall delay observed by the end-users. For example, Figure 3 shows a user in Japan who wants to access a page in a Web site in US. The request will pass through several ISP gateways before reaching the original Web site in the US. Since gateways are likely to be the main bottlenecks and since there are many others factors along the Internet paths between the user and the origin that may contribute to delays, even if the response time of the source is close to zero, the end-user in Japan may observe large delays.

One obvious way to eliminate network delays is by using a high speed dedicated line to deliver the contents without or reducing passing through Internet gateways. This solution sometimes is used by large companies to link their geographically dispersed offices and by server farms to ship content quickly between their content centers. However, it is clear that implementing such an approach as a general solution would be prohibitively expensive.

An alternative approach for reducing the network delay is to use intelligent caching techniques; i.e., deploying many cheap mirror servers, proxies, and other intermediary short-term storage spaces in the network and serving users from sources closer to them. This approach is commonly referred to as edge-based content delivery services and the architectures that provide content delivery services are referred to as edge-based content delivery networks (CDNs). Akamai (Akamai Technologies, 2001),and Adero (Adero Inc, 2001) are some of the companies that provide edge-based services.

Content Delivery Services

Several high-technology companies (Digital Island, 2001; Akamai Technologies, 2001; Adero Inc, 2001; CacheFlow Inc, 2001; InfoLibria Inc., 2001) are competing feverishly with each other to establish network infrastructures refereed to as a *content delivery networks (CDNs)*. The key technology underlying all CDNs is the deployment of network-wide caches which replicate the content held by the origin server in different parts of the network: front-end caches, proxy caches, edge caches, and so on. The basic premise of this architecture is that by replicating the HTML content, user requests for a specific content may be served from a cache that is in the network proximity of the user instead of routing it all the way to the origin server.

In CDNs, since the traffic is redirected intelligently to an appropriate replica, the system can be protected from traffic surges and the users can observe fast response times. Furthermore, this approach can not only eliminate network delays, but it can also be used to distribute the load on the servers more effectively. There are several advantages of this approach:

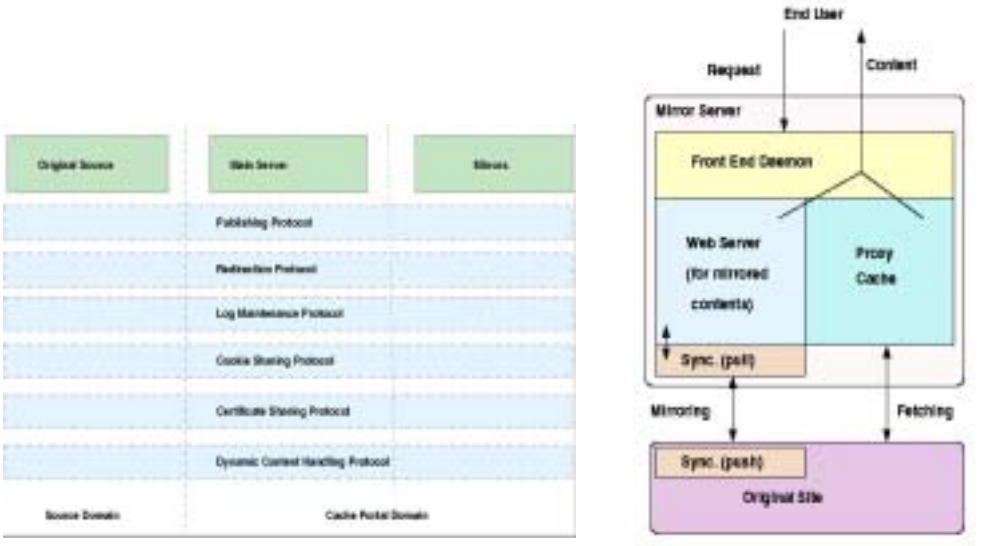
- User requests are satisfied in more responsive manner due to lower network latency.
- Since requests are not routed completely from the user site to the origin server, significant bandwidth savings can be potentially realized.
- Origin servers can be made more scalable due to load distribution. Not all requests need

to be served by the origin server; network caches participate in serving user requests and thereby distributing the load.

Of course these advantages are realized at the cost of additional complexity at the network architecture level. For example, new techniques and algorithms are needed to route/forward user requests to appropriate cache. Akamai (Akamai Technologies, 2001) enables caching of embedded objects in an HTML page and maintains multiple versions of the base HTML page (index, html) such that the HTML links to the embedded objects point to the cached copies. When a user request arrives at the origin server that has been *akamized,* an appropriate version of index, html is returned to the user to ensure that the embedded objects in the base HTML page are served from the Akamai caches that are close to the user site. In general, current architectures restrict themselves to the caching of static content (e.g., image data, video data, audio data, etc.) or content that is updated relatively infrequently. The origin server and the caches have to rely on manual or hard-wired approaches for propagating the updates to the caches in the latter case. We see that there are two major approaches for developing architectures for content delivery networks:

- *Network-level solutions:* In this case, content delivery services are built around existing network services, such as domain name servers (DNSs), IP multicasting, etc. The advantage of this approach is that it does not require a change in the existing Internet infrastructure and can be deployed relatively easily. Also, since in most cases the network protocols are already optimized to provide these services efficiently, these solutions are likely to work fast. However, since most existing network services are not built with integrated content delivery services in mind, it is not always possible achieve all desirable savings using this approach.
- *Application-level solutions:* In order to leverage all possible savings in content delivery, another approach is to bypass the services provided by the network protocols and develop an application-level solutions, such as application level multicasting (eg.,

Figure 4: (a) Content delivery architecture and (b) a mirror server in this architecture



FastForward networks, which is acquired by Inktomi (Inktomi, 2001)). These solutions rely on constant observations of network-level properties and responding to the corresponding changes using application-level protocols. Since providers of these solutions can finetune their application logic to the specific needs of a given content delivery service, this approach can be optimized to provide different savings (such as, bandwidth utilization, response time, prioritized delivery of content) as needed and can provide significant benefits. The main challenges with this approach, however, are to be able to observe the network state accurately and constantly and to deploy a costly Internet-wide application infrastructure.

Since they provide greater flexibility and more options, in this chapter, we mostly focus on the application-level architectures for content delivery services. Figure 4(a) shows the three entities involved in a content delivery architecture: the original source, a potential main server (or redirection server), and mirror servers. The original site is the e-commerce site maintained by the customer; i.e., the e-commerce business owner. The main redirection server is where the redirection decisions are given (note that in some architectures the redirection decision may be given in a truly distributed fashion). In effect, this server is similar in function to a domain name server, except that it functions at the application level, instead of functioning at the IP-level. The mirror servers, on the other hand, are the servers in which content is replicated/cached. In Figure 4(a), we show the architecture of a generic mirror server, which integrates a Web server and a proxy server:

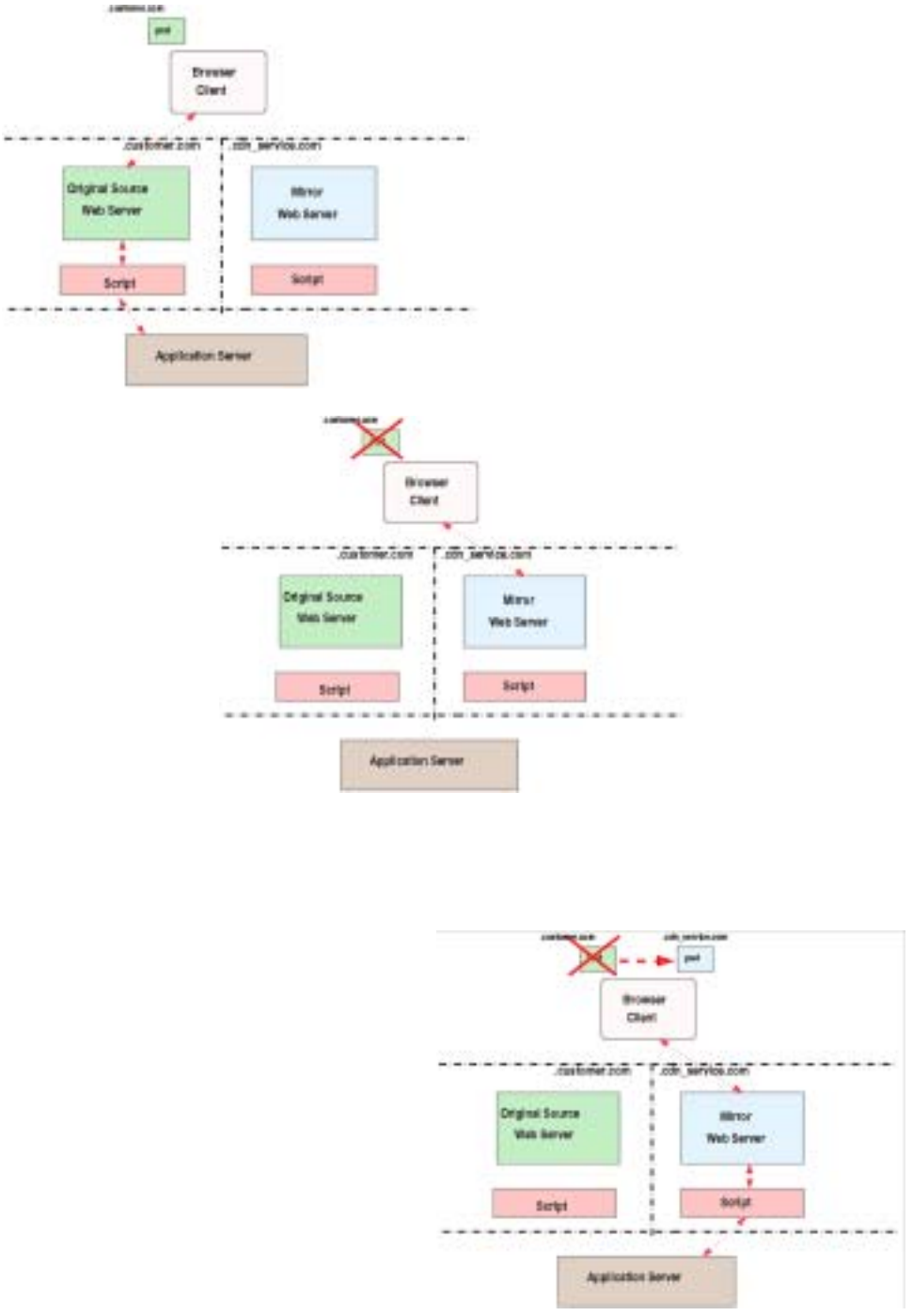
- The web server component serves the data that has been published by the original web server in advance (before the request is received by the mirror server).
- When a user request arrives to the mirror server, it is first processed by the web server and the proxy cache. If the content is found in either component, the content is delivered. If the content is not found, it has to be fetched from the original web site and copied into the proxy cache if the attribute of the content is not specified as "non-cacheable".
- If the server is too overloaded to perform these tasks, it can redirect traffic to other mirror sites.

Various protocols are required for these three entities to cooperate. Figure 4(a) lists these protocols. Note that different service providers implement these protocols differently. Consequently, achieving interoperability between providers requires agreements on protocols and development of common infrastructures (Content Bridge, 2001).

Publishing Protocol

Publishing protocol enables the content available at the original site to be replicated and distributed to the mirror sites. Depending on the architecture, the protocol can be push- or pull-based, can be object or site dependent, and can be synchronous or asynchronous: (1) In a push-based protocol, the source decides when and which objects to push to the mirrors. In a pull-based protocol, on the other hand, the mirrors identify when their utilization drops below a threshold and then request new objects from the original source. Note that it is also possible that mirrors will act as simple, *transparent* caches, which store only those objects that pass through them. In such a case, there is no publishing protocol required. (2) In an object-based protocol, the granularity of the publishing decision is at the object-level; i.e., the access rate to each object is evaluated separately and only those objects that are likely to be requested at a particular mirror will be published to that mirror server. In a site-based protocol, however, the entire site is mirrored. (3) In a synchronous protocol, publication is

Figure 5: *Problem with cookies in multi-domain CDNs: (a) The original site writes a cookie into the clients browser, (b) when the client is directed to a mirror, the cookie available at the client can no longer be accessed, therefore (c) while the client is being redirected to a mirror, the system must create a copy of the existing cookie at the client*



performed at, potentially regular intervals, at all mirrors; whereas, in an asynchronous protocol, publication decisions between the original server and each mirror are given separately.

The publishing protocol is the only protocol that is not directly involved in servicing user requests. The other protocols, shown in Figure 4, are all utilized in request-processing time, for

1. capturing the initial user request,
2. choosing the most suitable server for the current user/server/network configuration, and
3. delivering the content to the user.

Next, we discuss these protocols and their roles in the processing of user requests.

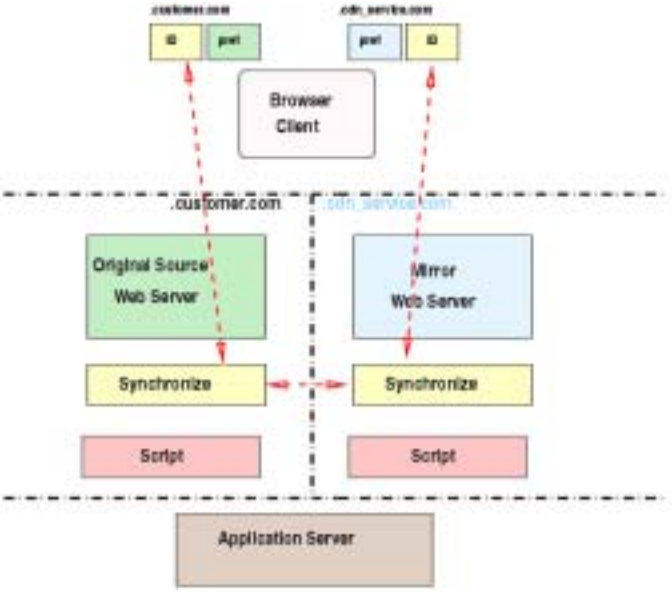
Cookie and Certificate Sharing Protocols

Most user requests arrive to the original source and, if necessary, they are directed to the appropriate mirror servers. Some user requests, though, may directly arrive to mirror servers. Therefore, both the original source and the mirror servers must be capable of capturing and processing user requests. There are two aspects of this task:

- masquerading as the original source while capturing the user (transaction) requests at the mirrors/caches and
- running the application logic and accessing/modifying the underlying data in the databases on the behalf of the original source.

Masquerading as the original source may require the mirror site to authenticate itself as the original source (i.e., have the same certificates). Furthermore, for state-dependent inputs (e.g., user history or preferences), mirrors should have access to state information (i.e., cookies that are placed into the memory space of client browsers by servers for enabling stateful browsing) maintained by other mirror servers or the original source.

Figure 6: Handling cookies, in multi-domain CDNs, without modifying the application programs



However, cookies can not be read by any domain except the ones which set them in the first place.

In DNS-based solutions, where all the replicas are seen as if they are under the same domain, this task does not cause any problems. However, if mirror servers have their own domain names, they require special attention (Figure 5). One way to share cookies across servers is shown in Figure 6:

1. Assign a unique ID to the user *the first time* user accesses the source and exchange this ID along with *the first redirection message.*
2. Use this ID to synchronize the cookie information between the original source and the mirror.
3. Performs these tasks without modifying existing applications:
   - intercept inputs and outputs through a synchronization module that sits between the web server and the application server, and
   - hide this process from the clients through the *server rewrite* option provided by web servers.

The cookie synchronization module manages this task by keeping a cookie information-base that contain the cookie information generated by the original site. Although, keeping the cookie data may be costly, it enables dynamic content caching with no modification on the application semantics. Furthermore, once the initial IDs are synchronized, irrespective of how many redirections are performed between mirrors and the original source, the cookie information will always be fresh. At any point in time, if the original site chooses to discontinue the use of the service, then it can do so without loosing any client state information.

Running the applications of the original source can be done either by replicating the application semantics at the mirror site or by delegating the application execution to a separate application server (or the original source).

Replicating the application semantics not only requires replicating the application and the execution environment, but also replicating/distributing the input data and synchronizing the output data. In other words, this task is equivalent to distributed transaction processing task. Although it is possible to provide a solution at the web-content level, the more general case (implementing a distributed database + application environment) is generally beyond the scope of current CDNs. However, due to the increasing acceptance of the Java 2 Enterprise Edition (J2EE) (Java(tm) 2 Platform, 2001), a platform independent distributed application environment, by the application server vendors, this is becoming a relevant task.

An application server is a bundle of software on a server or group of servers that provides the business logic for an application program. An application server sits along with or between the web server and the backend. which represents the database and other legacy applications that reside on large servers and mainframes. The business logic, which is basically the application itself and which acts as the middleware glue between the web server and the backend systems, sits within the application server.

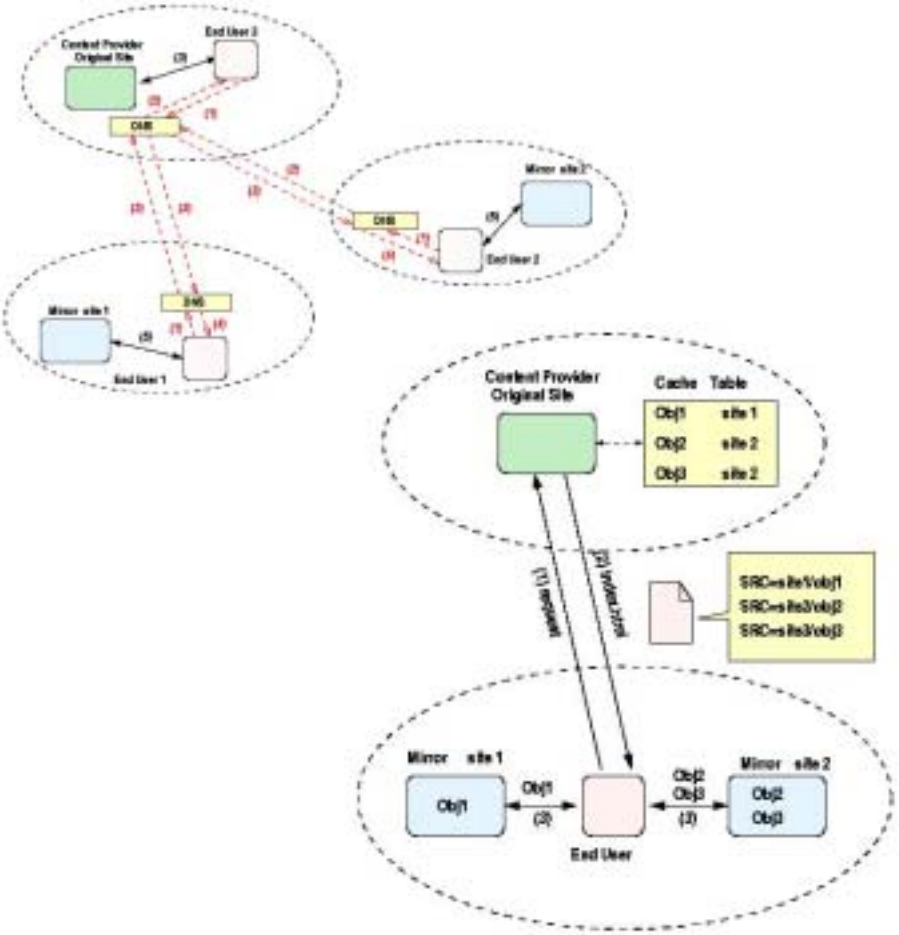
J2EE platform enables application builders to integrate pre-built application components into their products. Since many applications, such as those involved in e-commerce, contain common modules, independent software developers can save a great deal of time by building their applications on top of existing modules that already provide required functionality. This calls for a distributed architecture, where different modules can locate each other through directory services and can exchange information through messaging systems. In addition, for such a system to be practical, it has to support a container framework which will host modules that are independently created and transaction services to enable these independent modules perform business transactions. J2EE compliant application servers act

as containers for business logic/modules (enterprise Java beans) which provide their services to other modules and/or end-users. Note that J2EE compliant application servers provide the necessary framework for a replication environment, where application along with the data that they run on can be replicated at the edges. The resulting replicated application architecture enables dynamic load balancing and removes the single points of failure.

JXTA (Project JXTA, 2001) is another recent technology that can be used for developing distributed, interoperable, peer-to-peer applications. It includes the protocols for finding peers on dynamically changing networks, to share content with any peer within the network, to monitor peer activities remotely, and to securely communicate with peers. Although JXTA lacks many essential protocols required for facilitating the development of replicated/distributed applications, it provides some of the basic building blocks that would be useful in creating a CDN with many independent peer mirror servers.

J2EE is widely accepted by most major technology vendors (including IBM, SUN, BEA, and Oracle). A related technology, Microsoft's .NET strategy (Microsoft, 2001), uses an application server built using proprietary technology. But, in its essence, it also aims at hosting distributed services that can be integrated within other products.

Figure 7: Content delivery: (a) DNS redirection and (b) embedded object redirection



Note that, whichever underlying technology is utilized, delivering web content through a replicated/distributed application architecture will need to deal with dynamically changing data and application semantics. We will concentrate on the issues arising due to dynamicity of data in Section 3.

Redirection Protocol

A redirection protocol implements a policy that assigns user requests to most appropriate servers. As we mentioned earlier, it is possible to implement redirection at the network or application levels, each with its own advantages and disadvantages. Note that there may be more than one redirection policy used for different e-commerce systems, therefore a redirection protocol should be flexible enough to accommodate all existing redirection policies and be extendible to capture future redirection policies.

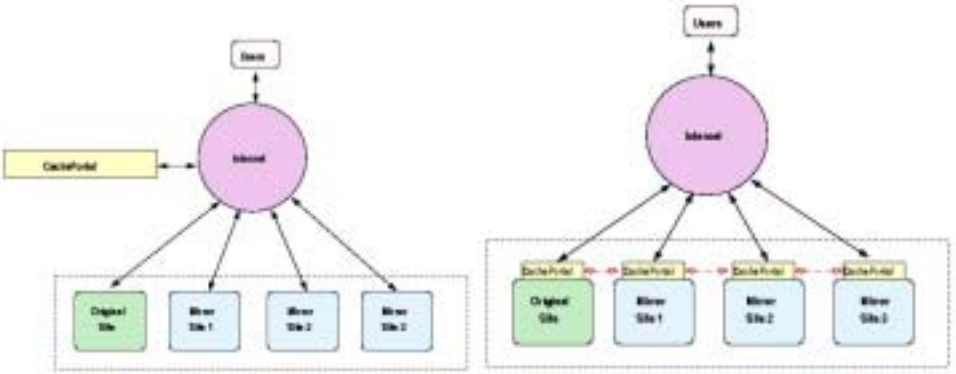
There are two ways that a request redirection service can be implemented: domain name server (DNS) redirection and application level redirection.

In DNS Redirection, the DNS at the original web site determines the mirror site closest to the end user based on his/her IP address and redirects the user to that mirror site. In Figure 7(a), end users I and 2, who are far from the content providers original server are redirected to local mirror servers, whereas the end user 3 gets the content from the original server. Since DNS redirection applies to all requests the same way, object-based solutions, where different objects or object types are redirected differently, can not be implemented.

In application-level solutions can be implemented in various ways. In this approach shown in Figure 7(b), all page requests are directed to the original site. Given a request, the original site checks the user's location based on the IP address associated with the request and then, it finds the closest mirror sites *(Site 1* and *Site2* in this example) containing the objects embedded in the requested page. The system then rewrites the HTML content of the Web page by specifying these most suitable object sources. When the browser parses this customized page, it learns that it has to go these servers to fetch the objects; i.e, in this example, the browser contacts *Site* 1 to fetch the object *Obj1* and *Site2* to fetch objects *Obj2* and *0bj3*.

Although DNS redirection has a very low overhead, it has the main disadvantage that it can not differentiate between semantics of different requests (e.g., media versus HTML page) as well as capabilities of the servers that are using the same domain name. Conse-

Figure 8: (a) Content delivery through central coordination and (b) through distributed decision making



quently, irrespective of what type of content (media, text, or stream) is requested, all servers must be ready to serve them. Furthermore, all content must be present at all servers. Therefore, this approach does not lend itself into intelligent load balancing. Since dynamic content delivery is very sensitive to the load on the servers, however, this approach can not be preferred in e-commerce systems.

Figure 9: Redirection process, alternative I

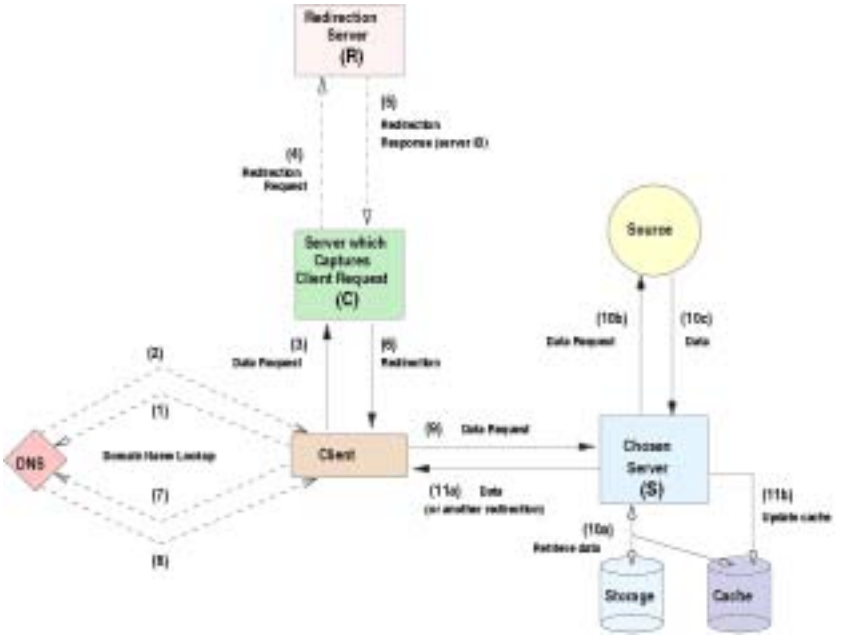
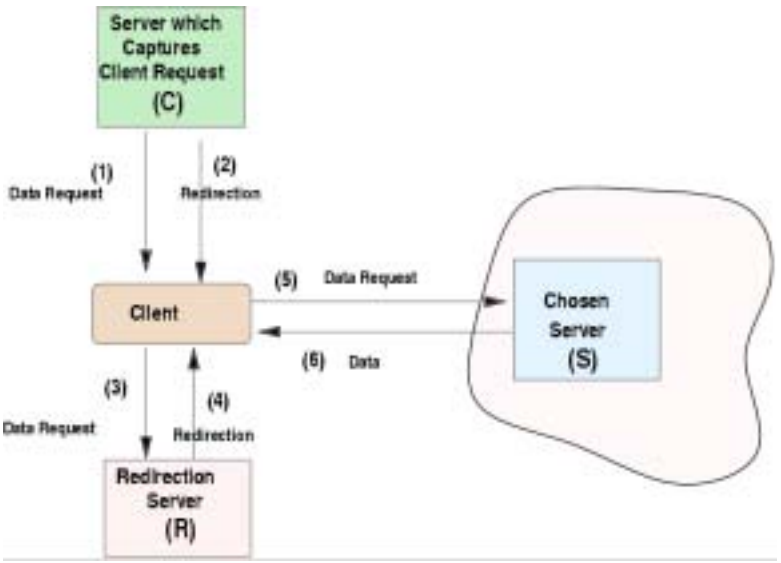


Figure 10: Redirection process, alternative 2 (simplified graph)



Note that it is also possible to use various hybrid approaches. Akamai (Akamai Technologies, 2001), for instance is using an hybrid of the two approaches depicted in figures 7(a) and (b). But, whichever implementation approach is chosen, the main task of the redirection is, given a user request, to identify the most suitable server for the current server and network status.

The most appropriate mirror server for a given user request can be identified either using a centralized coordinator (a dedicated redirection server) or allowing distributed decision making (each server performs redirection independently).

In Figure 8(a), there are several mirror servers coordinated by a main server. When a particular server experiences a request rate higher than its capability threshold, it requests the central redirection server to allocate one or more mirror servers to handle its traffic.

In Figure 8(b), each mirror server software is installed to each server. When a particular serer experiences a request rate higher than its capability threshold, it checks the availability at the participating servers and determines one or more servers to serve its contents.

Note, however, that even when we use the centralized approach, there can be *more than one* central server, distributing the redirection load. In fact, the central server(s) can broadcast the redirection information to all mirrors, in a sense converging to a distributed architecture, shown in Figure 8(b)). In addition, a central redirection server can either act as a passive directory server (Figure *9)* or an active redirection agent (Figure 10):

- As shown in Figure *9,* the server which captures the user request can communicate with the redirection server to choose the most suitable server for a particular request. Note that, in this figure, arrow *(4)* and *(5)* denote a subprotocol between the first server and the redirection server, which act as a directory server in this case.
- Alternatively, as shown in Figure 10, the first server can redirect the request to the redirection server and let this central server to choose the best content server and redirect the request to it.

The disadvantage of the second approach is that the client is involved in the redirection process twice. This reduces the transparency of the redirection. Furthermore, this approach is likely to cause two additional DNS lookups by the client: one to locate the redirection server and the other to locate the new content server. In contrast, in the first option, the user browser is involved only in the final redirection (i.e., only once). Furthermore, since the first option lends itself better to caching of redirection information at the servers, it can further reduce the overall response time as well as the load on the redirection server.

Figure 11: Four different URL streams mapped to three different pages: the parameter (cookie, GET, or POST parameter) ID *is not a caching key*

The redirection information can be declared *permanent* (i.e. cacheable) or *temporary* (non-cacheable). Depending on whether we want ISP proxies and browser caches to contribute to the redirection process, we may choose either permanent or temporary redirection: The advantage of the permanent redirection is that future requests of the same nature will be automatically redirected. The disadvantage is that since the ISP proxies are also involved in the future redirection processes, the CDN looses the complete control of the redirection (hence load distribution) process. Therefore, it is better to use either temporary redirection or permanent redirection with a relatively short expiration date. Since, most browsers may not recognize temporary redirection, the second option is preferred. The expiration duration is based on how fast the network and server conditions change and how much load balancing we would like to perform.

Log Maintenance Protocol

For a redirection protocol to identify the best suitable content server for a given request, it is important that the server and network status are known as accurately as possible. Similarly, for the publication mechanism to correctly identify which objects to replicate to which servers (and when), statistics and projections about the object access rates, delivery costs, and resource availabilities must be available.

Such information is collected throughout the content delivery architecture (servers, proxies, network, and clients) and shared to enable the accuracy of the content delivery decisions. A log maintenance protocol is responsible with the sharing of such information across the many components of the architecture.

Dynamic Content Handling Protocol

When indexing the dynamically created web pages, a cache has to consider not only the URL string, but also the cookies and request parameters (i.e., HTTP GET and POST parameters), as these are used in the creation of the page content. Hence, a caching key consists of three types of information contained within an HTTP request (we use the Apache (Apache, 2001) environment variable convention to describe these):

- the HTTP_HOST string,
- a list of (cookie,value) pairs (from the HTTRCOOKIE environment variable),
- a list of (GET parameter name,value) pairs (from the QUERYSTRING), and
- a list of (POST parameter name,value) pairs (from the HTTP message body).

Note that, given an HTTP request, different GET, POST, or cookie parameters may have different effects on caching. Some parameters may need to be used as keys/indexes in the cache, whereas some other may not (Figure 11). Therefore, the parameters that have to be used in indexing pages have to be declared in advance and, unlike caches for static content, dynamic content caches must be implemented in a way to use these keys for indexing.

The architecture described so far works very well for static content; that is content which does not change often or whose change rate is predictable. When the content published into the mirror server or cached into the proxy cache can change unpredictably, however, the risk of serving stale content arises. In order to prevent this, it is necessary to utilize a protocol which can handle dynamic content. In the next section, we will focus on this and other challenges introduced by dynamically generated content.

IMPACT OF DYNAMIC CONTENT ON CONTENT DELIVERY ARCHITECTURES

As can be seen from the emergence of J2EE and .NET technologies, in the space of web and Internet technologies, there is currently a shift toward service-centric architectures. In particular, many brick-and-mortar companies are reinventing themselves to provide services over the web. Web servers in this context are referred to as *e-commerce* servers. A typical e-commerce server architecture consists of three major components: a database management system (DBMS) which maintains information pertaining to the service, an application server (AS) which encodes business logic pertaining to the organization, and a web server (WS) which provides the web-based interface between the users and the e-commerce provider. The application server can use a combination of the server side technologies, such as

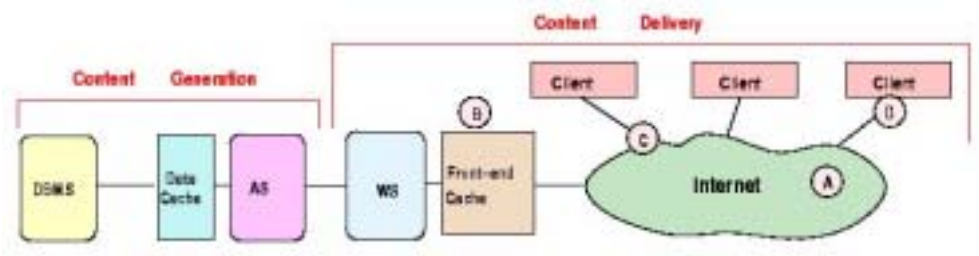
- the Java Servlet technology (Java(TM) Servlet Technology, 2001), which enables Java application components to be downloaded into the application server;
- JavaServer Pages (JSP) (JavaServer Pages(TM) Technology, 2001) or Active Server Pages (ASP) (Microsoft ASP.NET, 2001), which use tags and scripts to encapsulate the application logic , within the page itself;
- JavaBeans (JavaBeans(TM), 2001), Enterprise JavaBeans (Enterprise JavaBeans(TM) Technology, 2001), or ActiveX (MicrosoftCOM Technologies, 2001) software component architectures that provide automatic support for services, such as transactions, security, and database connectivity;

to implement application logic. In contrast to traditional web architectures, user requests in this case invoke appropriate program scripts in the application server which in turn issue queries to the underlying DBMS to dynamically generate and construct HTML responses and pages. Since executing application programs and accessing DBMSs may require significant time and other resources, it may be more advantageous to cache application results in a *result cache* (Labrinidis and Roussopoulos, 2000; Oracle9i web cache, 2001), instead of caching the data used by the applications in a data cache (Oracle9i data cache, 2001).

The key difference in this case is that database driven HTML content is inherently *dynamic* and the main problem that arises in caching such content is to ensure its *freshness*. In particular, if we blindly enable dynamic content caching we run the risk of users viewing stale data specially when the corresponding data-elements in the underlying DBMS are updated. This is a significant problem since the DBMS typically stores inventory, catalog, and pricing information which gets updated relatively frequently. As the number of e-commerce sites increases, there is a critical need to develop the next generation of CDN architecture which would enable dynamic content caching. Currently, most dynamically generated HTML pages are tagged as non-cacheable or expire-immediately. This means that every user request to dynamically generated HTML pages must be served from the origin server.

Several solutions are beginning to emerge in both research laboratories *(Challenger, Dantzig, and Iyengar, 1998;* Challenger, Iyengar, and Dantzig, 1999; Levy, Iyengar, Song, and Dias, 1999; Douglis, Haro, and Rabinovich, 1999; Smith, Acharya, Yang, and Zhu, 1999) and commercial arena (Persistence Software Systems Inc, 2001; Zembu Inc., 2001; Oracle Corporation, 2001). In this section, we identify the technical challenges that must be overcome to enable dynamic content caching. We also describe architectural issues that arises with regard to the serving dynamically created pages.

Figure 12: A typical e-commerce site (WS: Web server; AS: Application server; DS:Database server)



Overview of Dynamic Content Delivery Architectures

Figure 12 shows an overview of a typical web page delivery mechanism for web sites with backend systems, such as database management systems. In a standard configuration, there are a set of web/application servers that are load balanced using a traffic balancer, such as Cisco LocalDirector (Cisco, 2001). In addition to the web servers, e-commerce sites utilize database management systems (DBMSs) to maintain business related data, such as prices, descriptions, and quantities of products. When a user accesses the web site, the request and its associated parameters, such as the product name and model number, are passed to an application server. The application server performs necessary computation to identify what kind of data it needs from the database and, then, sends appropriate queries to the database. After the database returns the query results to the application server, the application uses these to prepare a web page and passes the result page to the web server, which then sends it to the user.

In contrast to a dynamically generated page, a static page, i.e., a page which has not been generated on demand, can be served to a user in a variety of ways. In particular, it can be placed in

- a proxy cache (Figure 12(A)),
- a web server front-end cache (as in reverse proxy caching, Figure 12(B)),
- an edge cache (i.e. a cache close to users and operated by content delivery services, Figure 12(C)), or
- a user side cache (i.e. user site proxy cache or browser cache, Figure 12(D))

for future use. Note, however, that the application servers, databases, web servers, and caches are independent components. Furthermore, there is no efficient mechanism to make database content changes to be reflected to the cached pages. Since most e-commerce applications are sensitive to the freshness of the information provided to the clients, most application servers have to mark dynamically generated web pages as *non -cacheable* **or make** them expire immediately. Consequently, subsequent requests to dynamically generated web pages with the same content result in repeated computation in the backend systems (application and database servers) as well as the network roundtrip latency between the user and the e-commerce site.

In general, a dynamically created page can be described as a function of the underlying application logic, user parameters, information contained within cookies, data contain within databases, and other extemal data. Although it is true that any of these can change during the lifetime of a cached web page, rendering the page *stale,* it is also true that

- application logic does not change very often and when it changes it is easy to detect;
- user parameters can change from a request to another request; however, in general many user requests may share the same *(popular)* parameter values;

- cookie information can also change from a request to another request; however, in general many requests may share the same *(popular)* cookie parameter values;
- external data (filesystem + network) may change unpredictably and undetectably; however, most e-commerce web applications do not use such external data; and
- database contents can change; however such changes can be detected.

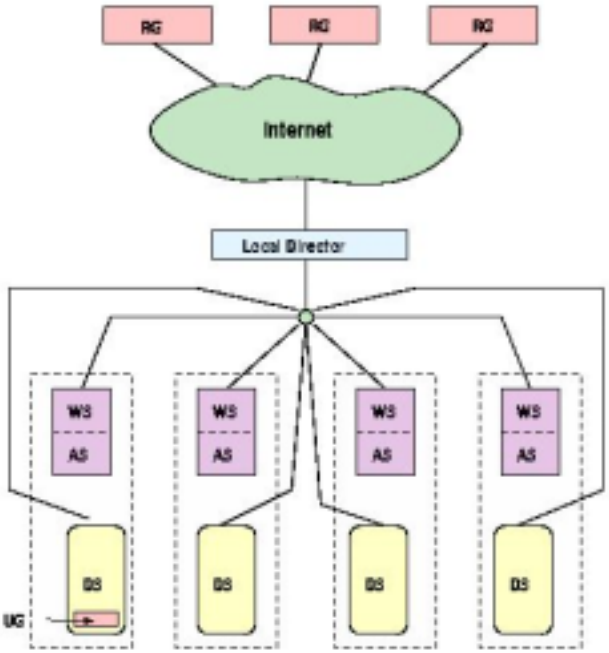
Therefore, in most cases, it is unnecessary and very inefficient to mark all dynamically created pages as *noncacheable,* as it is mostly done in current systems. There are various ways in which current systems are trying to tackle this problem. In some e-business applications, frequently accessed pages, such as catalog pages, are pre-generated and placed in the web server. However, when the data on the database changes, the changes are not immediately propagated to the web server. One way to increase the probability that the web pages are fresh is to periodically refresh the pages through the web server (for example, Oracle9i web cache provides a mechanism for time-based refreshing of the web pages in the cache) (Oracle9i web cache, 2001). However, this results in a significant amount of unnecessary computation overhead at the web server, the application server, and the databases. Furthermore, even with such a periodic refresh rate, web pages in the cache can not be guaranteed to be up-to-date.

Since caches designed to handle static content are not useful for database-driven web content, e-commerce sites have to use other mechanisms to achieve scalability. Below, we describe three approaches to e-commerce site scalability.

Configuration I

Figure 13 shows the standard configuration, where there are a set of web/application servers that are load balanced using a traffic balancer, such as Cisco LocalDirector. Such a

Figure 13: Configuration I (replication); RGs are the clients (requests generators) and UG is the database where the updates are registered



configuration enables a web site to partition its load among multiple web servers, therefore achieving higher scalability. Note, however, that since pages delivered by e-commerce sites are database dependent (i.e., puts computation burden on a database management system) replicating only the web servers is not enough for scaling up the entire architecture. We also need to make sure that the underlying database does not become a bottleneck. Therefore, in this configuration, database servers are also replicated along with the web servers. Note that this architecture has the advantage of being very simple; however, it has two major shortcomings. First of all, since it does not allow caching of dynamically generated content, it still requires redundant computation when clients have similar requests. Secondly, it is generally very costly to keep multiple database synchronized in an update-intensive environment.

Configuration II

Figure 14 shows an altemative configuration that tries to address the two shortcomings of the first configuration. As before, a set of web/application servers are placed behind a load balancing unit. In this configuration, however, there is only one DBMS serving all web servers. Each web server, on the other hand, has a middle tier database cache to prevent the load on the actual DBMS from growing too fast. Oracle 8i provides a middle-tier data cache (Oracle9i data cache, 2001), which serves this purpose. A similar product, Dynamai (Persistence Software Systems Inc, 2001), is provided by Persistence software. Since it uses middle-tier database caches *(DCaches),* this option reduces the redundant accesses to the DBMS; however, it can not reduce the redundancy arising from the web server and application server computations. Furthermore, although it does not incur database replication overheads, ensuring the currency of the caches requires a heavy database-cache synchronization overhead.

Figure 14: Configuration II (middle-tier data caching)

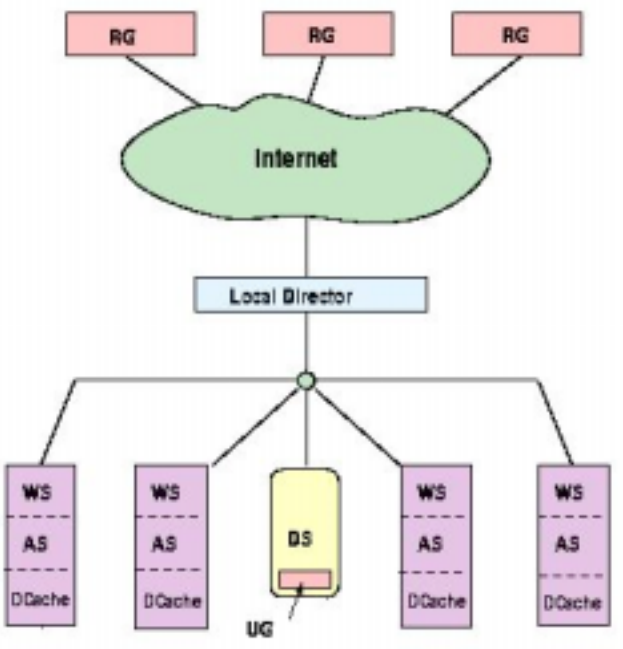
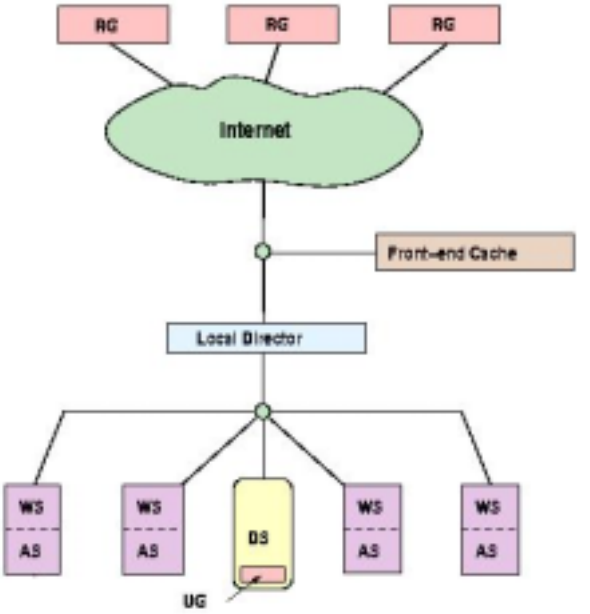


Figure 15: Configuration III (Web caching)



Configuration III

Finally, Figure 15 shows the configuration, where a dynamic-web-content cache sits in front of the load balancer to reduce the total number of web requests reaching the web server farm. In this configuration, there is only one database management server. Hence, there is no data replication overhead. Also, since there is no middle-tier data cache, there is also no database-cache synchronization overhead. The redundancy is reduced at all three (WS, AS, and DS) levels.

Note that, in this configuration, in order to deal with dynamicity (i.e., changes in the database) an additional mechanism that will reflect the changes in the database into the web caches is required. One way to achieve invalidation is to embed, into the database, update sensitive triggers which generate invalidation messages when certain changes to the underlying data occurs. The effectiveness of this approach, however, depends on the trigger management capabilities (such as tuple versus table level trigger activation and join-based trigger conditions) of the underlying database. More importantly, it puts heavy trigger management burden on the database. In addition, since the invalidation process depends on the requests that are cached, the database management system must also store a table of these pages. Finally, since the trigger management would be handled by the database management system, the invalidator would not have control over the invalidation process to guarantee timely invalidation.

Another way to overcome the shortcomings of the trigger-based approach is to use materialized views whenever they are available. In this approach, one would define a materialized view for each query type and then use triggers on these materialized views. Although this approach could increase the expressive power of the triggers, it would not solve the efficiency problems. Instead, it would increase the load on the DBMS by imposing unnecessary view management costs.

Network Appliance NetCache4.O (Network Appliance Inc, 2001) supports an extended HTTP protocol, which enables demand-based *ejection* of cached web pages. Similarly, recently, as part of its new application server, Oracle9i (Oracle9i web cache, 2001), Oracle announced a web cache that is capable of storing dynamically generated pages. In order to deal with dynamicity, Oracle9i allows for time-based, application-based, or trigger-based invalidation of the pages in the cache. However, to our knowledge, Oracle9i does not provide a mechanism through which updates in the underlying data can be used to identify which pages in the cache to be invalidated. Also, the use of triggers for this purpose is likely to be very inefficient and may introduce a very large overhead on the underlying DBMSs, defeating the original purpose. In addition, this approach would require changes in the original application program and/or database to accommodate triggers. Persistence software (Persistence Software Systems Inc, 2001) and IBM *(Challenger, Dantzig, and Iyengar, 1998;* Challenger, Iyengar, and Dantzig, 1999; Levy, Iyengar, Song, and Dias, 1999) adopted solutions where applications are finetuned for propagation of updates from applications to the caches. They also suffer from the fact that caching requires changes in existing applications

In (Candan, Li, Luo, Hsiung, and Agrawal, 2001), CachePortal, a system for intelligently managing dynamically generated web content stored in the caches and the web servers, is described. An invalidator, which observes the updates that are occurring in the database identifies and invalidates cached web pages that are affected by these updates. Note that this configuration has an associated overhead: the amount of database polling queries generated to achieve a better-quality finer-granularity invalidation. The polling queries can either be directed to the original database or, in order to reduce the load on the DBMS, to a middle-tier data cache maintained by the invalidator. This solution works with the most popular components in the industry (Oracle DBMS and BEA WebLogic web and application server).

Enabling Caching and Mirroring in Dynamic Content Delivery Architectures

Caching of dynamically created pages requires a protocol, which combines the HTML *expires* tag and an *invalidation* mechanism. Although the expiration information can be used by all caches/mirrors, the invalidation works only with compliant caches/mirrors. Therefore, it is essential to push invalidation as closer to the end-users as possible. For time-sensitive material (material that users should not access after expiration) that reside at the non-compliant caches/mirrors, the *expires* value should be set to 0. Compliant caches/mirrors also must be able to validate requests for non-compliant caches/mirrors.

In this section we concentrate on the architectural issues for enabling caching of dynamic content. This involves reusing of the unchanged material whenever possible (i.e., incremental updates), sharing of dynamic material among applicable users, prefetching/precomputation (i.e., anticipation of changes), and invalidation.

Reusing unchanged material requires considering the web content that can be updated at various levels: the structure of an entire site or a portion of a single HTML page can change. On the other hand, due to the design of the web browsers, updates are visible to end-users only at the page level. That is whether the entire structure of a site or a small portion of a single web page changes, users observe changes only one page at a time. Therefore, existing cache/mirror managers work at the page level; i.e., they cache/mirror pages. This is consistent with the access granularity of the web browsers. Furthermore, this approach works well with changes at the page or higher levels: if the structure of a site changes, we can reflect this by

removing irrelevant pages, inserting new ones, and keeping the unchanged pages.

The page level management of caches/mirrors, on the other hand, does not work well with subpage level changes. If a single line in a page gets updated, it is wasteful to remove the old page and replace it with a new one. Instead of sending an entire page to a receiver, it is more effective (in terms of network resources) to send just a delta(URL, change location, change length, new material) and let the receiver perform a page rewrite *(Banga, Douglis, and Rabinovich, 2001)*. Recently, Oracle and Akamai proposed a new standard called Edge Site Includes (ESI) which can be used to describe which parts of a page is dynamically generated and which parts are static (ESI, 2001). Each part can be cached as independent entities in the caches and the page can be assembled into a single page at the edge. This allows the static content to be cached and delivered by Akamai's static content delivery network. The dynamic portion of the page, on the other hand, is to be recomputed as required.

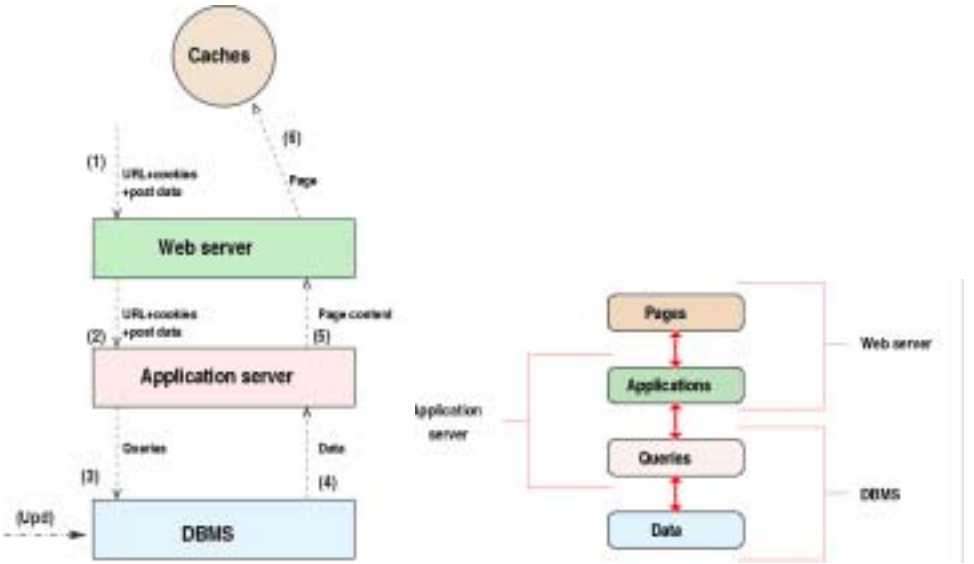
The concept of independently caching the fragments of a web page and assembling them dynamically has significant advantages. First of all, the load on the application server is reduced. The origin server now needs to generate only the non cacheable parts in each page. Another advantage of ESI is the reduction of the load on the network. ESI markup language also provides for environment variables and conditional inclusion, thereby allowing personalization of content at the edges. ESI also allows for an explicit invalidation protocol. As we will discuss soon, explicit invalidation is necessary for caching dynamically generated web content.

Prefetching and Precomputing can be used for improving performance. This requires anticipating the updates and prefetch the relevant data, precomputing the relevant results, and disseminating them to compliant end-points in advance and/or *validate* them

- either on demand (validation initiated by a request from the end-points, or
- by a special validation message from the source to the compliant end-points.

This, however, requires understanding of application semantics, user preferences, and the nature of the data to discover what updates may be done in the near future.

Figure 16: (a) Data flow in a database driven web site and (b) how different entities are related to each other and which web site components are aware of them



Chutney Technologies (Chutney Technologies, 2001) provides a PreLoader software that benefits from precomputing and caching. PreLoader assumes that the original content is augmented with special Chutney tags, as with ESI tags. PreLoader employs a predictive *least-likely to be used* cache management strategy to maximize the utilization of the cache.

Invalidation mechanisms mark appropriate dynamically created pages cacheable, detect changes in the database that may render previously created pages invalid, and invalidate cache content that may be obsolete due to changes.

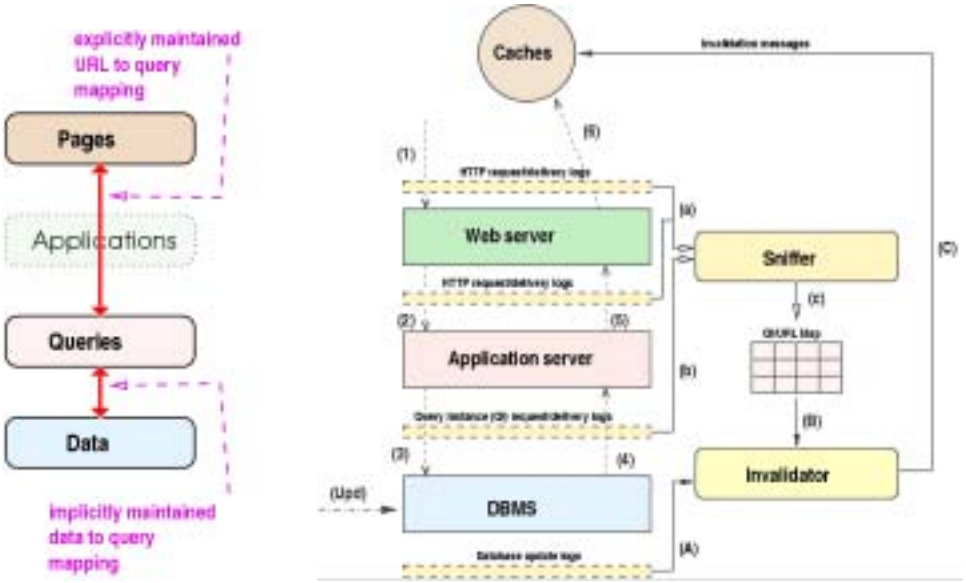
The first major challenge an invalidation mechanism faces is to create a mapping among the cached web pages and the underlying data elements (Figure 16(a)). Figure 16(b) shows the dependencies between the four entities (pages, applications, queries, and data) involved in the creation of dynamic content. As shown in this figure, knowledge about these four entities are distributed on three different servers (web server, application server, and the database management server). Consequently, it is not straightforward to create an efficient mapping between the data and the corresponding pages.

The second major challenge is that timely web content delivery is a critical task for e-commerce sites and that any dynamic content cache manager must be very efficient (i.e., should not impose additional burden on the content delivery process), robust (i.e., should not increase the failure probability of the site), independent (i.e., should be outside of the web server, application server, and the DBMS to enable the use of products from different vendors), and non-invasive (i.e., should not require alteration of existing applications or special tailoring of new applications).

CachePortal (Candan, Li, Luo, Hsiung, and Agrawal, 2001) addresses these two challenges efficiently and effectively. Figure 17(a) shows the main idea behind the CachePortal solution:

- Instead of trying to find the mapping between all four entities in Figure 17(a), CachePortal divides the mapping problem into two: it finds (1) the mapping between

Figure 17: Invalidation-based dynamic content cache management: (a) the bi-level management of page to data mapping and (b) the server independent architecture for managing the bi-level mappings



web pages and queries that are used for generating

This bi-layered approach enables the division of the problem into two components: *sniffing* or mapping the relationship between the web pages and the underlying queries and, once the database is updated, *invalidating* the web content dependent on queries that are affected by this update. Therefore, CachePortal uses an architecture (Figure 17(b)), which consists of two independent components, a *sniffer,* which collects information about user requests and an *invalidator,* which removes cached pages that are affected by updates to the underlying data.

The sniffer/invalidator sits on a separate machine, which fetches the logs from the appropriate servers at regular intervals. Consequently, as shown in this Figure 17(b), the sniffer/invalidator architecture does not interrupt or alter the web request/database update processes. It also does not require changes in the servers or applications. Instead it relies on three logs (the HTTP request/delivery log, the query instance/delivery log, and the database update logs) to extract all the relevant information. Arrows **(a)-(c)** show the sniffer query instance/URL map generation process and arrows **(A)-(C)** show the cache content invalidation process. These two processes are complementary to each other; yet they are asynchronous.

At the time of the writing, various commercial caching and invalidation solutions exist. Xcache (Xcache, 2001) and Spider Cache (SpiderSoftware, 2001) both provide solutions based on triggers and manual specification of web content and the underlying data. No automated invalidation function is supported. Javlin (Object Design, 2001) and Chutney (Chutney Technologies, 2001) provide middleware level cache/pre-fetch solutions, which lie between application servers and underlying DBMS or file systems. Again, no real automated invalidation function is supported by these solutions. Major application server vendors, such as IBM WebSphere (WebSphere Software Platform, 2001), BEA WebLogic (BEA Systems, 2001), SUN/Netscape I-planet (iPlanet, 2001), and Oracle Application Server (Oracle Application Server, 2001) focus on EJB (Enterprise Java Bean) and JTA (Java Transaction API *(Java(TM)Transaction API, 2001))* level caching for high performance computing purpose. Currently, these commercial solutions do not have intelligent invalidation functions either.

Impact of Dynamic Content on the Selection of the Mirror Server

Assuming that we can cache dynamic content at network-wide caches, in order to provide content delivery services, we need to develop a mechanism through which end-user requests are directed to the most appropriate cache/mirror server. As we mentioned earlier, one major characteristic of e-commerce content is that they are usually small (~4k); and hence, the network delay observed by the end-users are less sensitive to the network delays compared with large media objects, unless the delivery path crosses (mostly logical) geographic location barriers. In contrast, however, dynamic content is extremely sensitive to the loads in the servers. The reason for this sensitivity is that, it usually takes three servers, a database server, an application server, and a web server, to generate and deliver those pages; and the underlying database and application servers are generally not very scalable and they become bottleneck before the web servers and the network.

Therefore, since the characteristics of the requirements for dynamic content delivery is different from delivering static media objects, we see that the content delivery networks need to employ suitable approaches depending on their data load. In particular, we see that

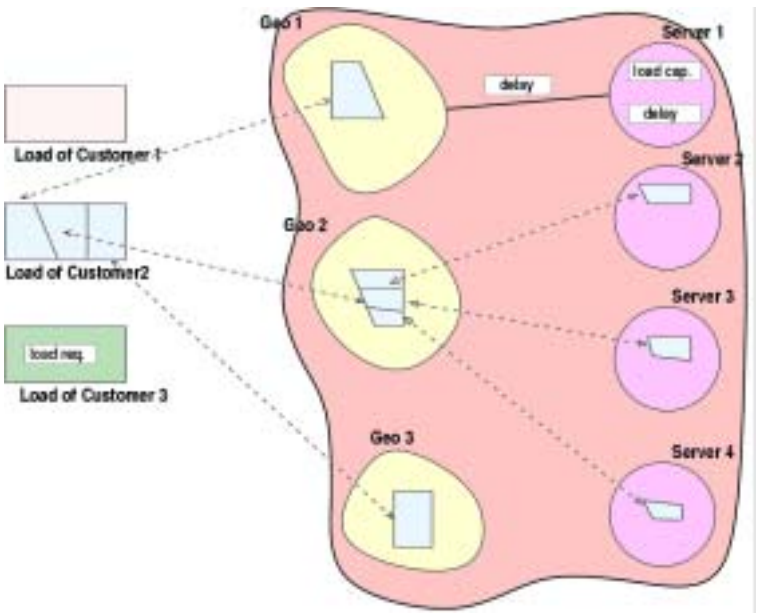
it may be desirable to distribute end-user requests across geographic boundaries, if the penalty paid by the additional delay is less then the gain observed by the reduced load on the system. We also note that, since the mirroring of dynamically generated content is not as straight forward as mirroring of the static content, in quickly changing environments, we may need to use servers located in remote geographic regions, if no server in a given region contains the required content.

However, when the load is distributed across network boundaries, we can no longer use pure load balancing solutions, as the network delay across the boundaries also becomes important (Figure 18). Therefore, it is essential to improve the observed performance of a dynamic-content delivery network by assigning the end-user requests to servers intelligently, using the following characteristics of CDNs:

- the type, size, and resource requirements of the published web content (in terms of both storage requirements at the mirror site and transmission characteristics from mirror to the clients),
- the load requirement (in terms of the requests generated by their clients per second),
- the geographic distribution of their load requirement (where are their clients at a given time of the day), and
- the performance guarantees that they require (such as the response time observed by their end-users).

Most importantly, these characteristics, along with the network characteristics, can change during the day as the usage pattems of the end-users shift with time of the day and the geographic location. Therefore, a static solution (such as a predetermined optimal content placement strategy) is not sufficient. Instead, it is necessary to dynamically adjust the client-to-server assignment.

Figure 18: Load distribution process for dynamic content delivery networks: The load of customers of a CDN comes from different geographic locations; however, a static solution where each geographic location has its own set of servers may not be acceptable.



RELATED WORK

Various content delivery networks (CDNs) are currently in operation. These include Adero (Adero Inc, 2001), Akamai (Akamai Technologies, 2001), Digital Island (Digital Island, 2001), MirrorImage (Mirror Image Internet, Inc., 2001) and others. Although each one of these services are using more or less different technologies, they all aim to utilize a set of web-based network elements (or servers) to achieve efficient delivery of web content. Currently, all of these CDNs are mainly focused on the delivery of static web content. (Johnson, Carr, Day, Kaashoek, 2001) provides a comparison of two popular CDNs (Akamai and Digital Island) and concludes that the performance of CDNs is more or less the same. It also suggests that the goal of a CDN should be to choose a *reasonably* good server, while avoiding *unreasonably* bad ones, which in fact justifies the use of a heuristic algorithm. (Paul and Fei, 2000), on the other hand, provides concrete evidence that shows that a distributed architecture of coordinated caches perform consistently better (in terms of hit ratio, *response time,* freshness, and *load balancing).* These results justify the choice of using a centralized load assignment heuristic.

Other related works include (Heddaya and Mirdad, 1997; Heddaya, Mirdad, and Yates, 1997), where authors propose a diffusion-based caching protocol that achieves load-balancing, (Korupolu and Dahlin, 1999) which uses meta-information in the cache-hierarchy to improve the hit ratio of the caches, (Tewari, Dahlin, Vin, and Kay, 1999) which evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems, and (Carter and Crovella, 1999) which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection.

CONCLUSIONS

In this chapter, we described the state of art of e-commerce acceleration services. We point out their disadvantages, including failure to handle dynamically generated web content. More specifically, we addressed two questions faced by e-commerce acceleration systems: (1) what changes the characteristics of the e-commerce systems require in the popular content delivery architectures and (2) what is the impact of end-to-end (Internet+server) scalability requirements of e-commerce systems on e-commerce server software design. Finally, we introduced an architecture for integrating Internet services, business logic, and database technologies, for improving end-to-end scalability of e-commerce systems.

REFERENCES

Zona Research, 2001. http://www.zonaresearch.com/.

Digital Island, 2001. Ltd. http://www.digitalisland.com/.

Exodus Communications, 2001. http://www.exodus.com/.

Mirror Image Internet, Inc., 2001. instaDelivery Internet Services. http://www.mirrorimage.com/.

Akamai Technologies, 2001. http://www.akamai.com.

Adero Inc, 2001. http://www.adero.com/.

CacheFlow Inc, 2001. http://www.cacheflow.com/.

InfoLibria Inc., 2001. http://www.infolibria.com/.

Inktomi, 2001. http://www.inktomi.com/.

Content Bridge, 2001. Global Content Distribution Architecture. http://www.content-bridge.com.

Java(tm) 2 Platform, 2001. Enterprise Edition. http://java.sun.com/j2ee.
Project JXTA, 2001. http://www.jxta.org/.
Microsoft, 2001. .Net. http://www.microsoft.com/servers/evaluation/overview/net.asp.
Apache, 2001. HHTP Server Project. http://httpd.apache.org/.
Java(TM) Servlet Technology, 2001. http://java.sun.conilproducts/servlet.
JavaServer Pages(TM) Technology, 2001. http://java.sun.conilproducts/jsp.
Microsoft ASP.NET, 2001. http://www.asp.net.
JavaBeans(TM), 2001. http://java.sun.comlproducts/javabeans.
Enterprise JavaBeans(TM) Technology, 2001. http://java.sun.com/products/ejb.
MicrosoftCOM Technologies, 2001. ActiveX Control. http://www.microsoft.com/com/tech/activex.asp.
A. Labrinidis and N. Roussopoulos, 2000. WebView Materialization. In Proceedings of the ACM SIGMOD, pp 367-378, 2000.
Oracle9i web cache, 2001. http://www.oracle.com/ip/deploy/ias/ caching/index.html?web_caching. html.
Oracle9i data cache, 2001. http://www.oracle.com/ip/deploy/ias/caching/index.html?database_ caching. html
Network Appliance Inc., 2001. http://www.networkappliance.com/.
Jim Challenger, Paul Dantzig, and Arun Iyengar, 1998. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. In *Proceedings of ACM/IEEE Supercomputing '98,* Orlando, Florida, November 1998.
Jim Challenger, Arun Iyengar, and Paul Dantzig, 1999. Scalable System for Consistently Caching Dynamic Web Data. In *Proceedings of the IEEE INFOCOM'99,* pp.294-303*,* New York, New York, March 1999. IEEE.
Eric Levy, Arun Iyengar, Junehwa Song, and Daniel Dias, 1999. Design and Performance of a Web Server Accelerator. In *Proceedings of the IEEE INFOCOM* '99, pp. 135-143, New York, New York, March 1999. IEEE.
Fred Douglis, Antonio Haro, and Michael Rabinovich, 1997. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In *Proceedings of USENIX Symposium on Internet Technologies and Systems,* 1997.
Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu, 1999. Exploiting Result Equivalence in Caching Dynamic Web Content. In *Proceedings of USENIX Symposium on Internet Technologies and Systems,* 1999.
Persistence Software Systems Inc, 2001. http://www.dynamai.com
Zembu Inc, 2001. http://www.zembu.com.
Oracle Corporation, 2001. http://www.oracle.com/.
Cisco, 2001. LocalDirector 400 Series http://www.cisco.com/warp/public/cc/pd/cxsr/400/.
K. Se1çuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal, 2001. Enabling Dynamic Content Caching for Database-Driven Web Sites. In *Proceedings of the 2001 ACM SIGMOD* , Santa Barbara, CA, USA, May 2001. ACM.
G. Banga, F. Douglis, and M. Rabinovich, 1997. Optimistic Deltas for WWW Latency Reduction. In *Proceedings of the USENIX Technical Conference,* 1997.
ESI, 2001. Accelerating E-Business Applications. http://www.esi.org/.
Chutney Technologies, 2001. http://www.chutneytech.com.
Xcache, 2001. The Cache Management Solution For Fast, Scalable Web Sites. http://www.xcache.com.
SpiderSoftware, 2001. http://www.spidercache.com.
Object Design, 2001. Javlin Product Section. http://www.objectdesign.com/htm/

javlin_prod.asp.
WebSphere Software Platform, 2001. http://www.ibm.com/websphere.
BEA Systems, 2001. http://www.bea.com.
iPlanet, 2001. http://www.iplanet.com.
Oracle Application Server, 2001. http://www.oracle.com/ip/deploy/ias.
Java(TM)Transaction API, 2001. http://java.sun.com/products/jta.
K.L. Johnson, J.F. Carr, M.S. Day, and M.F. Kaashoek, 2000. The Measured Performance of Content Distribution Networks. Computer Communications 24(2), pages 202-206 (2001)
S. Paul and Z. Fei, 2000. Distributed Caching with Centralized Control. In 5th *International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.
A. Heddaya and S. Mirdad, 1997. WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. In ICDCS, 1997.
A. Heddaya, S. Mirdad, and D. Yates, 1997. Diffusion-based Caching: WebWave. In NLANR Web Caching Workshop, 9-10 June 1997.
M.R. Korupolu anf M. Dahlin, 1999. Coordinated Placement and Replacement for Large-Scale Distributed Caches. In IEEE Workshop on Internet Applications, pages 62—71, 1999.
R. Tewari, M. Dahlin, H.M. Vin, and J.S. Kay, 1999. Beyond Hierarchies: Design Considerations for Distribued Caching on the Internet. In ICDCS, pages 273-285, 1999.
R.L. Carter and M.E. Crovella, 1999. On the Network Impact of Dynamic Server Selection. In Computer Networks, volume 31, pages 2529—2558, 1999.

Section IV: Web-based Distributed Data Mining