World Wide Wait: A Study of Internet Scalability and Cache-Based Approaches to Alleviate It

Anindya Datta • Kaushik Dutta • Helen Thomas • Debra VanderMeer
Chutney Technologies, Inc., Atlanta, Georgia 30326
Florida International University, Miami, Florida 33199
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
Chutney Technologies, Inc., Atlanta, Georgia 30326
adatta@chutneytech.com • kaushik.dutta@fiu.edu • hthomas@andrew.cmu.edu • dvanderm@chutneytech.com

The Internet is growing rapidly in terms of both use and infrastructure. Unfortunately, demand is outpacing the capacity of the infrastructure, as evidenced by unacceptably long response times. To support current load and further growth, we must address this problem. Several caching strategies have been proposed in the literature; many have been implemented to improve the quality of service on the Web. In this paper, we identify the main causes of delay on the Web, and provide a review of the various caching strategies employed to mitigate these delays. We also survey the application of Operations Research/Management Science (OR/MS) techniques to caching on the Web. Finally, we identify several open OR/MS research problems related to Web caching.

1. Introduction

1.1. The World Wide Wait Problem

The Internet is experiencing phenomenal growth on two fronts simultaneously: (1) the size of the content delivered, and (2) the number of users served. This tremendous growth is clogging the World Wide Web (WWW), both at the origin websites and throughout the network, making it difficult to maintain acceptable response times for end users. The resulting increased response times experienced by end users is a phenomenon better known as the “World Wide Wait.” According to Zona Research, if the response time of a Web page is more than 8 seconds, 30% of the users will abandon the site. Unfortunately, the average response time for many popular sites at peak load is large enough to see the “Zona effect”—users leaving for competing outlets, resulting in a loss of revenue. So, although we are seeing tremendous growth of the Internet across several dimensions, we are experiencing a serious problem—the inability of the WWW to scale. Without new technologies to improve Internet scalability, the Internet will not be able to support the type of growth we have seen to date. This paper is about one such key class of technology that is being widely deployed across the broad Internet spectrum.

It is important to note that the inability to scale is a problem not only for the Internet infrastructure, but also for individual websites. In other words, the scale problem permeates both network and server-side issues. One common server-side problem occurs when a large number of users simultaneously log on to a site. This problem is known as the flash crowd problem, extreme cases of which are referred to as the denial of service phenomenon. Given the bursty nature of Internet traffic (Crovella and Bestvros 1996),
most websites face the flash crowd problem daily. The problem, of course, is that users expect sites to respond to their requests within an acceptable time period, not only when there is just one user on the site, but also when there are thousands or even potentially millions of users.

Consider the case of the 2000 U.S. Presidential Election. Suppose a user, let’s call him Frank, prefers to see the election results through his favorite news website. Frank was interested in the results, so he requested updated information every few moments. Naturally, he expected the response of his favorite news site to be as fast as it was on nonelection days. However, he found the response of the site to be much slower than normal. Why was this? The number of visitors to the site was many times higher than usual due to the importance of the event. For example, on election night, CBSNews.com saw the number of unique visitors to its site grow from 103,000 to 341,000. From November 5, 2000, to November 7, 2000, the number of unique visitors to MSNBC.com spiked from a normal load of 747,000 to 2.64 million. To compound the problem, many of these users frequently reloaded pages to get current results, so the click rate of each of these users was much higher than the usual click rate. Thus, the site needed to scale to support large numbers of simultaneous “Franks,” even when all of these “Franks” were clicking much more frequently than usual. However, according to information from an Internet reporting firm, Keynote Systems, Inc., MSNBC.com’s average response time between 6:00 p.m. and 9:00 p.m. EST on election day was 32.54 seconds, with 59% of the time taken in page loading. According to Keynote Systems, Inc., on November 8, 2000, at times it took almost two minutes to download the New York Times main page.

Clearly, scalability is a critical factor for the success of business, commerce, and entertainment across the Internet, and it is a problem that attracts, perhaps, the most amount of attention among Internet and WWW infrastructure issues. One frequently chosen approach to improving scalability is to employ caching, where content is generated once and kept for future reuse. When a user requests content that was previously requested by some other user, the content can be served to the second user directly from the cache. This cache may reside at various places across the Internet. To see this, consider the structure of the Internet as shown in Figure 1.

The hierarchy consists of end-user systems, which connect to local Internet service providers (ISPs). Local ISPs, in turn, connect to regional ISPs, which connect to national backbone providers (NBPs). The NBPs are interconnected, either through network access points (NAPs) or private peering relationships. For additional details on the Internet structure, refer to Datta et al. (2003). Caching can occur at many different locations throughout this structure, e.g., at the end-user systems, the local or regional ISPs, and so on.

For example, in the case of Frank, the election results change, but not every second. Perhaps the results change every 15 minutes. With a cache, we could serve the same content to all the “Franks” who click within this 15-minute interval. Additionally, all the “Franks” who are in the same geographic area may have some common interest in the election, say, for local election results. If all these Franks living in the same area are connected to the Internet through the same few local ISPs, then it would be quite logical to get the content once from the news site and cache it at the local ISP. In this scenario, a user need not
Figure 1  The Internet Structure

Key
- CP  Content Provider
- EU  End User
- ISP  Internet Service Provider
- NAP  Network Access Point
- NBP  National Backbone Provider

Go all the way to the news site itself until the content changes; rather, the request can be directly served from the cache of the local ISP. The existence of such a cache would reduce the delay in response, while simultaneously increasing the number of concurrent users the site can support with acceptable levels of quality of service (QoS), and reducing the bandwidth on the network.

It turns out that caching can be employed across multiple dimensions to accelerate and scale the Internet and the WWW. For instance, one can consider caching along the dimension of the Internet structural hierarchy, an example of which was provided in the previous paragraph—we demonstrated the utility of caching at the local ISP. One can also consider caching along the content dimension—it is possible to cache various kinds and granularities of content such as rich content (e.g., a graphic object), complete Hypertext Markup Language (HTML) pages, or even partial page fragments. Yet another dimension along which caching makes sense is the functional dimension—one can cache the results of the domain name server (DNS) lookup function, for instance. Indeed, caching is currently employed ubiquitously throughout the Internet and is one of the most successful acceleration technologies employed. The purpose of this paper is threefold:

1. To introduce the reader to the World Wide Wait or the “scale” problem of the Internet. We do this by performing a thorough analysis of the various delay-causing elements of the Internet.
2. To perform a comprehensive examination of the various caching strategies currently employed.
3. To expose readers to a number of open problems, the bulk of which tend to be optimization problems.

The rest of this paper is organized as follows. In §2, we discuss the causes of delay on the WWW. In §3, we discuss caching as a solution to these delays and present a taxonomy of Web-caching techniques. This is followed by a survey of Web-caching techniques in §4. In §5, we survey the OR/MS literature that is related to caching on the WWW, and in §6, we present several open OR/MS research problems related to WWW caching. Finally, we conclude the paper in §7.

2. The World Wide Wait: Causes

In this section, we discuss the sources of delay in the Internet and the WWW. Delayed response can
affect the growth of the Internet and e-commerce. According to a widely cited survey (Kehoe et al. 1999), 55% of online users mentioned “slow download” as a dissatisfying experience with online shopping. When such a dissatisfied user churns (i.e., departs without purchasing), the site loses revenue in sales and advertising. Zona Research (1999) concluded that $4.3 billion in online sales were lost in 1999 due to poor Internet connections. Their studies further indicated that improving the response time, even by a small amount, has a profound effect on increasing buy rates. The churn rate will drop to just 7% if the response time is less than 7 seconds. Thus, a 1-second reduction in network delay can decrease the churn rate by 2–4 times.

Although the reasons for the delayed responses are manifold, the root cause is the delay induced by the multitude of devices that comprise the WWW infrastructure. These delays are collectively referred to as device latency. Server-side devices (such as computers) and network devices (such as switches and routers) require processing time to complete the tasks involved in generating and serving Web content to users. Delay occurs at a particular device when it cannot serve incoming requests at a rate equal to the arrival rate of those requests, resulting in growing queues of requests and increased response time from the device.

To understand the device latency problem on the WWW requires an understanding of how Web requests are served. We explain this process next, followed by a discussion of the specific bottlenecks in this process.

2.1. How Web Requests Are Served
The basic interaction between a user and an e-business application is based on the request-response paradigm: The user sends a request to a site, and the site prepares a response and sends it to the user. This paradigm is embodied in the Hypertext Transport Protocol (HTTP) (W3C 1999). We briefly describe this paradigm here and refer the reader to Datta et al. (2003) for a more detailed explanation. The following steps are required to serve a Web request:

Requesting content. A user specifies a Universal Resource Locator (URL) (W3C Network Working Group 1999) in his browser that identifies the content he or she desires.

Identifying the website on the network. The browser submits a query to a DNS, which returns the internet protocol (IP) address (Information Sciences Institute 1981) corresponding to the specified URL’s WWW address.

Connecting to the website. A connection is established between the client (i.e., the user’s machine) and the server.

Submitting the HTTP request. The client submits the HTTP request to the server.

Generating the response. The server generates a response, i.e., an HTML stream, in one of two ways: (1) by serving predefined HTML, referred to as static content, or (2) by generating the HTML upon receipt of the request, referred to as dynamic content.

Retrieving the HTML response. The response page is delivered to the user through HTTP in two parts: (1) the HTML text that can be rendered by a browser, and (2) the embedded objects in the page (e.g., images, audio, video), which are also obtained through an HTTP connection.

2.2. Bottlenecks in Serving Web Requests
A typical Web interaction incurs four types of processing and data transmission overhead: (1) DNS lookup overhead, (2) Connection setup overhead, (3) Content generation overhead, and (4) Content delivery overhead. Each of these contributes to the cumulative end-to-end latency of e-business applications. We now elaborate on these delays.

DNS Lookup. The first task of the browser is mapping the URL to an IP address through a mechanism called DNS resolution. This requires a lookup in a DNS database on the network and the subsequent transmission of the mapping information from the DNS to the browser client, incurring processing latency on the DNS server and network latency in the transmission of the data to the client. A particular DNS server may be overloaded or unavailable, causing a client to contact (potentially) several servers to obtain an IP address mapping. The more servers a client must contact, the larger the delay incurred in the lookup process.
Connection Setup. Connection setup involves (1) finding a path through the network between the client and server, and (2) setting up a connection with the Web server to submit a request for content. Information exchange between clients and servers on the Web is performed using HTTP, typically using Transmission Control Protocol (TCP) as the underlying transport protocol. TCP connection establishment involves a round-trip delay that induces some latency observed by the user. Further delay is caused by “TCP slow start” (Jacobson 1998), which is essentially the time consumed by the several round trips that TCP uses to find the right transmission speed for a particular connection. Connection establishment is a significant component of the HTTP request-response interaction (Feldmann et al. 1999). Some improvements in connection establishment overhead have been realized via the introduction of persistent HTTP (Padmanabhan and Mogul 1995, Nielsen et al. 1997) in which a persistent TCP connection is kept open and reused to carry imminent future HTTP requests. However, there is still some overhead associated with maintaining connection state information at both end points.

Content Generation. After a client has established a connection with the website, the actual GET or POST request for content can be submitted. When a website receives an HTTP request, the content can be served in one of two ways: (1) static content (e.g., pregenerated HTML files or images) can be served directly from the Web server, or (2) a Web page can be assembled “on the fly” using dynamic scripting technologies.

In the case of static content, the Web server simply retrieves the needed content from memory or persistent storage and serves it to the user. This retrieval incurs a small amount of delay.

Dynamically generated pages, on the other hand, can add significant delays. In fact, a recent study by Morgan Stanley Dean Witter (2000) has shown that content generation delay constitutes 40% of the end-to-end delay on average. Typically, dynamic content generation applications are designed based on the Model View Controller (MVC) (Gamma et al. 1994) paradigm. In this paradigm, a top-level program (typically called a script) contains presentation logic, while the business logic is offloaded to business components, such as Enterprise Java Beans (EJB) (Sun Microsystems 2003) or Component Object Model (COM) objects (Microsoft 2002), which, in turn, may call out to various data sources and other applications (e.g., personalization services). Given that these scripts may be arbitrarily complex, a multitude of different types of delays can be introduced, including pure processing delays within the script and delays due to fetching content from various data sources (e.g., databases and remote data sources), as well as delays introduced in calls to other applications, e.g., Extensible Markup Language (XML) to HTML transformations via Extensible Stylesheet Language Transformations (XSLT).

Content Delivery. The content of a Web page (including all embedded objects) must physically travel the distance between the Web server and the client across network links. Typically, IP packets traverse approximately 17 hops from the server to the client on the Internet (Morgan Stanley Dean Witter 2000). Each hop incurs delay, especially when there is congestion on the network.

The size of the delivered content also has an impact on the network delay. In particular, rich content (e.g., audio, video) tends to be much larger than static content and, hence, requires more network resources. Table 1 (Morgan Stanley Dean Witter 2000) provides a comparison of the requirements of static and rich content in terms of size, bandwidth, and acceptable latency requirements.

To summarize, we have identified the following four primary sources of delay on the WWW: (1) DNS lookup, (2) connection setup, (3) content generation, and (4) content delivery. As we have shown, there are many additional factors that compound the latency problem. For instance, the size of the requested content can significantly increase the time required

<table>
<thead>
<tr>
<th>Metric</th>
<th>Static content</th>
<th>Rich media</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required bandwidth</td>
<td>56 Kbps</td>
<td>1.5 Mbps</td>
</tr>
<tr>
<td>Typical object size</td>
<td>90 KB</td>
<td>50 MB+</td>
</tr>
<tr>
<td>Bandwidth consumed</td>
<td>3 MB/hr</td>
<td>1 GB/hr</td>
</tr>
<tr>
<td>Maximum acceptable latency</td>
<td>8 seconds</td>
<td>0.4 seconds</td>
</tr>
</tbody>
</table>

Table 1 Rich Media vs. Static Content
to deliver the content. In addition, the number of requests has an impact on all four sources of identified delays. Given the rapid projected growth in Internet usage, this fact has serious scalability implications for e-business applications.

Having presented the primary sources of delay on the WWW, we now turn our attention to a discussion of caching solutions designed to mitigate these delays.

3. Web Caching: Cures for What Ails the Web

In this section, we first motivate the need for Web caching and then provide an overview of Web-caching techniques in the form of a taxonomy.

3.1. Motivation for Caching

One commonly used approach to address the World Wide Wait problem is to employ more infrastructure, i.e., more hardware, software, and bandwidth. Adding infrastructure is expensive—especially when considering the total cost of ownership for the infrastructure—and it tends to move bottlenecks rather than eliminate them. A wiser use of available capacity, both on websites and the Internet as a whole, would be to effectively route and store the content across the network. In this scenario, each time a client submits a request, the requested content need not necessarily be regenerated and sent across the full network path from the server to the client. Rather, if content can be cached at various points on the network and websites, requests can be served from these caches, reducing the amount of work (in both processing and data transmission) that must be done to satisfy a user’s request.

In general, caching refers to the storage of frequently requested information at a location closer to where it will be consumed, rather than at the original source where it is produced. Broadly speaking, basic Web caching works as follows: A user requests information and the website serves the requested content. Meanwhile, a caching system, at some point between the website and the user (or, perhaps, at one of the two endpoints), stores the content in anticipation of future need for that content. At a later time, when a request for the same content arrives, the content is served from cache.

Numerous caching techniques have been proposed to address the sources of latency described in §2. In the next section, we present a taxonomy of these caching techniques, which we will subsequently use in our survey of Web-caching techniques.

3.2. A Taxonomy of Web Caching

In general, any caching system must address the following three issues: (1) what to cache, (2) where to cache, and (3) when to cache. We briefly discuss each in the context of Web caching. A more detailed description of the taxonomy is available in Datta et al. (2003).

What to Cache. There are three broad types of elements that can be cached in Web-caching systems: (1) the mapping of server name to IP address can be cached to address delays associated with DNS lookup, (2) TCP connections can be cached to address delays associated with connection setup, and (3) content (e.g., HTML pages, image files) can be cached to address delays associated with content generation and content delivery. The remainder of this paper will primarily focus on content caching systems, since content caching has been the most active area in terms of research. Content can be further classified as static or dynamic. Both static and dynamic content can be further classified according to caching granularity (e.g., full pages, fragments).

Where to Cache. A Web cache usually resides in one of three locations: (1) at the client browser, (2) at a proxy somewhere in between the client and the website, or (3) at the website itself. We briefly discuss the latter two approaches, because they are the most relevant to this paper. A proxy cache resides somewhere between a client and a Web server. Proxy caches are usually configured to operate in one of two modes: forward or reverse. A forward proxy intercepts Web requests from clients. If a requested object is found in cache, the proxy will serve the object from cache. Otherwise, it will retrieve the requested object from the origin Web server and store it in cache. A forward proxy is typically located near a network gateway to reduce the bandwidth required across expensive
dedicated Internet connections. A reverse proxy is usually placed directly in front of a group of servers to reduce the number of requests that the servers must handle.

Caching at the website, referred to as server-side caching, involves placing caches at various points within the Web server site, e.g., at the database, application server, and/or Web server. The purpose of these caches is to offload the work of the various server-side components.

**When to Cache.** Cache management addresses the question of “when” to cache. We identify two important subquestions: (1) when are items placed in cache and (2) when are items removed from cache? To address the question of when to place items in cache, caching algorithms can be classified as either passive, i.e., loading an item into cache only as a result of a request for that item, or active, i.e., prefetching items in anticipation of future requests. Cache replacement and cache invalidation policies address the question of when to remove items from cache. Cache replacement policies determine which cached element to replace when the number of cached elements exceeds a certain size. Cache invalidation policies determine how the cache is notified when the source data changes.

Having presented our taxonomy of Web-caching techniques, we now use this taxonomy to discuss relevant work in Web caching.

### 4. Survey of Web-Caching Techniques

We present a brief survey of the Web-caching literature based on the taxonomy presented in §3.2. Due to space limitations, we discuss only content caching. For the interested reader, an extended version of this survey is available in Datta et al. (2003). We first discuss static content caching followed by dynamic content caching.

#### 4.1. Static Content Caching

Most static content (e.g., graphics, audio, video) has high bandwidth requirements when compared to dynamically generated content. For this reason, the work in static content caching has focused on reducing the network load through either browser caching (e.g., Cunha and Jaccoud 1997) or proxy caching. The majority of this work has concentrated on proxy caching, addressing issues such as cache replacement (e.g., Williams et al. 1996), cache invalidation (e.g., Liu and Cao 1997), and cooperative caching (e.g., Chankhunthod et al. 1996) in which multiple proxies support one another in serving requests for cached items. One area within proxy caching that has raised many challenging research problems, especially for the OR/MS community, is content distribution networks (CDNs). A CDN consists of a collection of servers that attempts to offload work from origin servers by delivering content on their behalf. The servers belonging to a CDN cache some or all of the origin server’s content and are located at different points around the network. When a request is served, the CDN attempts to locate a CDN server “close” to the client (e.g., in terms of geography, topology, or latency) to serve the request.

The tasks associated with the operation of a CDN can be classified into three broad areas: content distribution, request routing, and pricing. Content distribution refers to the activity of placing a CDN customer’s content objects on one or more servers. Most present-day CDNs distribute content based on a service-level agreement (SLA) between the customer and the CDN. The SLA specifies the desired level of QoS, which may be defined in terms of a variety of metrics such as the average delay experienced by end users or the amount of caching space allocated to a site on the CDN servers.

Request routing refers to the problem of directing client requests to a particular CDN server. Most CDNs use some form of DNS redirection (Barbir et al. 2002) in which the authoritative DNS server is controlled by the CDN. With this technique, the client sends a DNS query to the authoritative DNS server of the CDN, which replies with the IP address mapping for the CDN server that it deems most appropriate to serve the request. The authoritative DNS server also sends a time to live (TTL), which is the time period for which the mapping remains valid. The TTL is usually set to a small value so that the CDN can quickly change the mapping to accommodate changing load conditions in the network. The server selection decision is usually made based on proximity measures.
(e.g., latency, packet loss, hop counts) and/or server feedback information (e.g., central processing unit (CPU) load, number of connections) (Barbir et al. 2002).

Pricing refers to the activities associated with measurement and pricing of the services offered by the CDN. CDNs usually charge for two resources: storage, and bandwidth. Storage charges correspond to the distribution of content across the servers, while bandwidth charges typically correspond to the number of user requests serviced by the CDN. The specific pricing terms are stipulated in the SLA. Pricing models for CDN services vary, but typically fall into one of three types: (1) usage based (e.g., Akamai’s FreeFlow (Akamai Technologies 2003), (2) subscription based (e.g., Akamai’s EdgeSuite (Akamai Technologies 2003), or (3) a hybrid of usage and subscription based (e.g., Mirror Image’s Instance Content (Mirror Image Internet 2003), where overage fees are added to the subscription fee if a prespecified maximum usage level is exceeded.

4.2. Dynamic Content Caching
Two broad types of caching have been proposed for dynamic content: proxy caching, and server-side caching. The work in dynamic proxy caching includes page-level caching in which dynamically generated HTML pages are cached at proxies (e.g., Iyengar and Challenger 1997), and fragment-level caching in which fragments of dynamically generated content are cached at proxies (e.g., ESI Consortium 2001, Datta et al. 2002). Server-side caching encompasses a variety of different types of caching that is done at different locations within the site infrastructure. Server-side caching techniques include database caching, whereby database tables and/or query results are cached in main memory (e.g., TimesTen (TimesTen Software 2003)), page-level caching, whereby dynamic pages are cached at the Web server (e.g., SpiderCache (Spider-Software 2003), and fragment-level caching, whereby dynamic fragments are cached either within the application server process (e.g., WebLogic Server Cache Tags (BEA Systems 2003)) or externally (e.g., Datta et al. 2001).

Having provided a brief survey of caching techniques for the WWW, we now turn our attention to a discussion of the OR/MS work that has been done related to WWW caching.

5. Applications of OR/MS Techniques to WWW Caching
There has been significant application of OR/MS techniques to WWW caching. In this section, we survey this work in three distinct parts. First, we discuss the application of optimization techniques to address content delivery or network-related delays. This is followed by a discussion of the application of OR/MS techniques to the cache sizing problem. Finally, we include a discussion of the work in developing performance models of caching systems.

5.1. Optimization Models for Content Delivery
As previously mentioned, the tasks associated with the operation of a CDN can be classified into the areas of content distribution, request routing, and pricing. As it turns out, these areas each encompass a number of optimization problems, some of which have already been studied by OR/MS researchers. In particular, content distribution and pricing are areas that have received some attention in the OR/MS literature. The work in pricing is closely related to the work in differentiated QoS in caching systems. Work in each of these areas is discussed.

5.1.1. Content Distribution. From an OR/MS perspective, the content distribution problem refers to the problem of determining the optimal location of content in the network. This problem is similar, in many respects, to the well-known file and data allocation problems in distributed systems (Dowdy and Foster 1982, Apers 1988). Although the file and data allocation problems are similar in that they are both concerned with optimally distributing data across a set of network nodes, there are substantial differences between these two problems. In general, the file allocation problem is to optimally assign files to possibly different nodes in a network for query/update/execution purposes (Dowdy and Foster 1982). For a given request, if the file is local, processing occurs locally. If the file is remote, then the request is transferred to the remote system for processing. Most formulations of the file allocation problem are based on cost objectives (e.g., minimizing storage, query, update, or communication costs) or performance objectives (e.g., minimizing response time, maximizing system throughput).
Common constraints found in these formulations include storage capacity, response time, and availability constraints.

Distributed database systems are quite different from distributed file systems and, therefore, require different approaches to the allocation problem. One key difference is that the objects to be allocated are not known prior to allocation. It is usually not appropriate to allocate relations in a distributed database because users at different sites are often interested in different fragments of relations. For example, users in New York will usually be interested in the tuples corresponding to the New York area, whereas users in Tokyo will usually be interested in the tuples related to Tokyo. For this reason, fragments of relations are generally used as the unit of allocation. This allocation scheme can result in significant reductions in data transmission costs for query processing. Another key difference between file and database allocation is the access patterns. In the file allocation problem, the only transmissions required to combine data from different nodes are the transmissions from the source nodes to the result node where the result is computed. In distributed databases, query processing is much more complex in that it requires data transmissions between nodes where the fragments are allocated.

To capture these differences, the data allocation problem can be stated as a more general version of the file allocation problem: Given a set of queries and updates, their frequencies, and the set of network nodes where the results must be sent (1) determine the fragments to be allocated and (2) allocate these fragments and the operations on them to the nodes in the network such that a certain cost function is minimized (Apers 1988). Similar to the file allocation problem, the data allocation problem has been formulated using cost objectives (e.g., minimizing total transmission costs of queries and updates weighted by execution frequencies) and performance objectives (e.g., minimizing response time of queries and updates). Constraints in the data allocation problem include bandwidth, CPU utilization, and availability constraints.

Many of the same objectives and constraints found in the file and data allocation problems are also appropriate for the content distribution problem. There are, however, certain key differences between the content distribution problem and the file and data allocation problems that are worthy of discussion. First is the unit of allocation. With file allocation, the unit of allocation is limited to files, while with data allocation, the unit of allocation is fragments of relations. With content distribution, however, the unit of allocation can have more variation. For instance, content can be distributed at the cache level (often referred to as server replication), at the file or document level, or perhaps somewhere in between (e.g., a collection of related documents). Another key difference between content distribution and file and data allocation is access patterns. In the content distribution problem, access patterns are quite different from those found in distributed databases. There is no need for the allocation of query operations nor the transmission of partial results between nodes because Web requests operate at the file level. While access patterns in the content distribution problem would appear to be similar to those found in distributed file systems, this is not necessarily the case. Access patterns for Web content have been extensively studied in the literature, and have been shown to exhibit certain unique characteristics, such as high locality of reference (Breslau et al. 1999).

The OR/MS literature has addressed the content distribution problem primarily from the perspective of server replication, where the objective is to determine the location of a set of proxy caches. One approach is to formulate the problem as a dynamic program, which determines the optimal placement of $M$ proxies among $N$ potential sites assuming a single origin Web server and a linear topology (Li et al. 1998). The objective of the dynamic program is to minimize overall access latency. Access latency is expressed as the weighted distance between a given location (i.e., node) and the Web server, where distance is represented by a nonnegative real value that can be interpreted in a variety of ways (e.g., latency, hop count), and the weight represents the amount of traffic passing through the node. It was later recognized that one of the underlying cost functions in Li et al. (1998) carries a Monge structure (Woeginger 2000). By exploiting this structure, the authors derive a more efficient solution algorithm. In Li et al. (1999),
the authors follow a similar dynamic programming approach, except that a tree topology, rather than a linear topology, is assumed. With this topology, the root of the tree represents the origin Web server.

In Cidon et al. (2001), the problem of proxy location is also studied assuming a single origin server and a tree topology. One key difference is that Li et al. (1999) assume that the number of proxies needed is known in advance, whereas Cidon et al. (2001) determine the optimal number of proxies based on the storage cost. The authors present a dynamic programming formulation that minimizes overall cost, which is comprised of storage and communication costs. The authors also show that the same algorithm can be used to determine the optimal allocation of content objects across the tree. In Krishnan et al. (2000), the authors address the cache location problem with an emphasis on transparent en-route caches (TERCs). TERCs are located along the routes between clients and servers and are transparent to clients and servers. A TERC intercepts requests that pass through it and either satisfies the request or forwards the request along the regular routing path. The authors present optimal solutions for simple topologies such as line and ring. They also consider the problem of locating proxies for a single server in a tree topology and propose a dynamic programming algorithm for this case.

The use of a tree topology in the above-mentioned works is a simplification that allows the authors to develop optimal algorithms. This topology does not accurately represent the Internet topology (Qiu et al. 2001), however. The more general case of a graph is considered in Qiu et al. (2001), which addresses the problem of locating server replicas in a CDN. The authors formulate the problem as the NP-complete K-median problem, which minimizes the total cost of all requests. The cost of a request is defined as the distance between the two nodes, where the distance can reflect different types of performance metrics (e.g., latency, hop count). The authors present heuristics to solve the problem. Although the heuristics require estimates of client distance and load predictions, the authors show that these heuristics can provide solutions that are close to optimal.

Cache placement algorithms for CDNs based on client demand clustering are proposed in Barford et al. (2002). The unit of clustering considered is the autonomous system (AS), a network or group of networks sharing a common routing policy. Routing information is shared between ASs using the Border Gateway Protocol (BGP). The authors present a technique for identifying regions of client demand using best path information from BGP routing tables. This technique generates a tree of demand that forms the input for the cache placement algorithms. Two cache placement algorithms are presented: (1) a dynamic programming algorithm, and (2) a greedy algorithm. These algorithms minimize inter-AS traffic and client response time. Performance results indicate that these algorithms can greatly improve performance across random cache placement.

5.1.2. Differentiated QoS in Caching Systems.

The work in differentiated QoS in caching systems forms the basis for the OR/MS work in pricing for CDNs thus far. From an OR/MS perspective, the pricing problem generally refers to the problem of determining the pricing scheme such that the revenue of the CDN provider is maximized. The OR/MS work in pricing has focused on incorporating differentiated QoS within such a pricing scheme. For example, a CDN provider may offer multiple caching services, such as differential caching (i.e., different hit rates for different service classes), prefetching, or caching of dynamically generated content. Each of these services may provide a different level of QoS and, thus, can be priced such that revenue is maximized for the CDN.

Differential caching has been studied in Kelly et al. (1999) and Lu et al. (2001). This work focuses on delivering different cache hit ratios to different classes of users in the case of a single cache. For instance, Kelly et al. (1999) propose a biased cache replacement policy. The least-frequently-used (LFU) policy is generalized in such a way that allocation of shared cache space can be biased toward different classes of users based on the aggregate value of the user class. The authors focus primarily on Web servers as a user class, but also discuss a client-centered policy. The problem is formulated as an optimization problem in which a weighted hit ratio is maximized, and the weights represent the value received when a hit occurs. Simulation experiments indicate that the proposed policy is able to deliver higher aggregate value.
to servers than least-recently-used (LRU) or LFU and provides reasonable QoS.

One limitation of this work is that it does not guarantee the distance between performance levels of different classes. An approach that addresses this issue is presented in Lu et al. (2001). This work proposes a differentiated caching services architecture that allows different classes of content to receive different hit ratios. Performance differentiation is achieved by applying techniques from digital feedback control theory. An implementation based on Squid is described and used as the basis for a set of experiments. Performance results indicate that the approach can provide significantly better performance to the premium content classes.

The pricing problem itself has been recently studied. In particular, Hosanagar et al. (2002), address the problem of pricing caching services for differentiated QoS. Here, the authors present an optimal pricing strategy intended for Internet access providers (IAPs) and ISPs that offer multiple levels of caching service. The study analyzes the trade-offs between bandwidth savings and the potential to charge publishers for caching their content. For instance, the study examines whether an IAP or ISP should charge for a basic or low-quality caching service. In addition, the authors suggest an approach for determining the optimal allocation of cache space among the different service classes once the quality levels and pricing have been determined.

5.2. Optimization Models for Cache Sizing

Another problem that has been studied by the OR/MS community is the cache-sizing problem, i.e., determining the optimal cache size. Two approaches to the problem, both of which are based on economic factors, are presented in Kelly and Reeves (2000). These approaches determine the optimal cache-sizing policy based on expected work load and the cost of memory and bandwidth. The first approach is an idealized model that considers only the memory/bandwidth trade-off for a simple two-level caching hierarchy. In this model, the work load consists of independent references drawn from a known distribution, and the caches employ a perfect (LFU) policy. The perfect LFU policy maintains reference counts across evictions, whereas the standard LFU policy removes reference count information upon eviction. The second approach relaxes these assumptions and allows for arbitrary work loads, miss costs, and storage costs. Based on this model, the authors describe an efficient method for computing the optimal storage capacity for a single cache.

Another approach to the cache-sizing problem is presented in Mookerjee and Tan (2002). Here, an approximate analysis technique for the LRU policy is shown to be useful in selecting the appropriate cache size. This technique allows cache size to be determined in one of two ways: (1) a performance guarantee approach, or (2) an economic approach. In the former approach, the cache size can be chosen to achieve a specified level of performance in terms of expected delay and hit ratio. In the latter approach, the cache size is selected such that it minimizes the sum of the cache storage and delay costs. This method is appropriate when these costs can be estimated.

5.3. Performance Models of Caching Systems

An important aspect of Web caching (and caching in general) is performance analysis. Most studies on Web caching have used trace-driven simulation to analyze caching system performance (e.g., Breslau et al. 1999, Raunak et al. 2000). While trace-driven simulation has been widely used, there are certain drawbacks associated with this approach (Watson et al. 1999). For instance, the access patterns are determined by the workload of the selected trace. Furthermore, trace collection is typically a time-consuming process and usually requires a reduction phase in which unnecessary data is removed. To address these issues, Watson et al. (1999) propose a model-driven simulation approach to analyze proxy cache performance. In particular, a user access model is developed, which takes into account both frequency and recency of access. This model is used to simulate document requests, which are input to a model-driven simulation of Web-caching policy. The study measures the cache hit rate as cache size is varied and compares different replacement policies (e.g., LRU, LFU, size), including a policy proposed by the authors, least weighted usage (LWU). LWU, which incorporates both frequency and recency of use, is shown to perform the best.
There have been few attempts to develop mathematical analyses of caching policies. One of the first such studies is Bose and Cheng (2000). This work develops a network queueing model to examine the impact of a proxy cache on response times. The model considers a single cache in forward proxy mode and, thus, captures the impact on response times for users within an organization. The study examines the effect of several key parameters, including the hit rate, the arrival rate of requests, size of the documents requested, and speed of the proxy cache server. This study produced interesting results and insights. For example, the authors identify a crossover probability, which represents the minimum cache hit rate that is needed for a proxy cache to provide benefit. They show that this crossover probability decreases as the arrival rate of requests or document size increases. In addition, the results indicate that while increasing the power (e.g., increasing server rate) of the proxy cache server decreases response times, a diminishing rate of return phenomenon occurs and, thus, it may not be justifiable to make such upgrades to the server.

An interesting extension to this work is Cheng and Bose (2001), which uses a multiclass network queueing model to account for different types of requests (e.g., multimedia and nonmultimedia).

Another analytical performance model for a caching system is proposed in Mookerjee and Tan (2002), which presents a mathematical analysis of the LRU replacement policy in the context of browser caching. In this study, analytical expressions are derived for two performance metrics: expected delay per document access and hit ratio. An approximate method to compute these measures is also presented and shown to be extremely accurate. Finally, the LRU policy is compared to other policies that take into account document size and download latency. The results of this study indicate that no single policy dominates in terms of performance, and that it may be difficult to formulate a policy that considers both document size and delay.

In this section, we have focused on the problems related to WWW caching that have been addressed by OR/MS researchers thus far. In the next section, we identify several open OR/MS research problems related to WWW caching.

6. Open OR/MS Research Problems in WWW Caching

Our discussion of open problems includes optimization problems for content delivery, optimization problems for content generation, and performance models of caching systems.

6.1. Optimization Problems for Content Delivery

As discussed in §5, CDNs have presented OR/MS researchers with a number of optimization problems. In this section, we present open problems in the three areas previously discussed: (1) content distribution, (2) request routing, and (3) pricing. Before doing so, we first present a model of a CDN, which we will use to describe the problems and discuss possible formulations.

We use a variation of the model presented in Kangasharju et al. (2001), and model a CDN as a graph, \( G = (V, E) \), where \( V \) is the set of nodes and \( E \) is the set of edges. Each node represents an AS, having one CDN server with finite storage capacity. Each edge represents an inter-AS connection.

Let \( I \) denote the number of ASs in the network. Let \( S_i \) denote the storage capacity for AS \( i \), (in bytes), \( i \in \{1, 2, \ldots, I\} \), and \( \lambda_i \) denote the aggregate request rate for objects from AS \( i \). There are \( J \) objects, and an object \( j \) has a size of \( b_j \), \( j \in \{1, 2, \ldots, J\} \), bytes.

6.1.1. Content Distribution. As previously mentioned, CDNs typically distribute content according to the SLAs between the CDN and its customers. This method of distribution, however, may result in suboptimal placement of content. For example, if an SLA stipulates that a customer’s content is to be distributed to a large set of servers to cover a wide geographical region, then that customer’s content will be placed at these locations even though the demand for the content at all locations may not be sufficient to warrant such widespread distribution. Such placement of content may have debilitating effects on overall system performance, such as poor utilization of cache space, which may reduce cache hit ratios. Thus, what is needed, but has not been yet addressed, is a globally optimal method of content distribution.
Such a content distribution method should help CDN providers in establishing more realistic and accurate SLA terms for content distribution, thereby decreasing the likelihood that SLAs will be violated.

While the OR/MS community has addressed the content distribution problem, as previously mentioned, the problem has been primarily addressed from the perspective of server replication. The problem that remains open is object replication, i.e., determining the optimal placement of objects within a CDN. In the object replication problem, we assume that a CDN exists with a known network topology, and that each CDN server has a known storage capacity. Furthermore, we assume that the size and distribution of demand for objects is known. In other words, we are given a set of ASs, each AS having storage capacity $S_i$, and a set of objects, each object $j$ having size $b_j$. The problem is then to distribute the set of objects across the ASs such that some cost metric is minimized.

The decisions in this problem are which objects to store at which ASs. We can represent these decisions using binary variables $X_{ij}$, $i \in I$, $j \in J$, such that $X_{ij} = 1$ if object $j$ is stored at AS $i$, and $X_{ij} = 0$ otherwise. One possible objective function is to minimize the total cost to retrieve all objects. If we define $c_{ij}$ to be the cost to retrieve object $j$ from AS $i$, and $o_j$ to be the cost to retrieve object $j$ from the origin server (note that $c_{ij}$ and $o_j$ may represent cost in terms of latency, number of hops, and so on), then this objective can be formulated as

$$\sum_{i \in I} \sum_{j \in J} c_{ij} X_{ij} + o_j(1 - X_{ij}),$$

(1)

where the first term represents the cost to retrieve the objects stored at the ASs and the second term represents the cost to retrieve the objects not stored at any AS. Multiplying by the aggregate request rate for objects at AS $i$ ($\lambda_i$) and then summing across all objects and all ASs gives the total cost to retrieve all objects. A set of constraints in this problem would be the storage space available at each AS, which can be expressed as

$$\sum_{j \in J} b_j X_{ij} \leq S_i, i = 1, \ldots, I.$$

It is worth mentioning that the above formulation is quite simplistic and represents one possible way to formulate the content distribution problem. Our intent is to stimulate the reader’s interest in the problem, rather than to provide a complex formulation. For example, one important aspect of content distribution that is not addressed by this formulation is content updates. Incorporating aspects such as content updates obviously requires more complex formulations. Note that the file and data allocation literature (e.g., Dowdy and Foster 1982, Apers 1988) previously mentioned may be useful in developing such alternative formulations for this problem.

We have identified one published work that addresses the object replication problem. In Kangasharju et al. (2001) the authors propose a model that considers replicating objects from a set of origin servers. The authors formulate the problem as a combinatorial optimization problem. The optimization problem is to replicate objects so that when clients retrieve objects from the closest CDN server that has the requested object, the average number of nodes traversed is minimized. The authors show that the problem is NP-complete, present heuristics to solve the problem, and evaluate the performance of these heuristics. Their performance results indicate that the best performing heuristic is one in which all servers cooperate. This work provides a good starting point for researchers in studying the object replication problem. There are many ways in which this model can be augmented and/or extended. For instance, alternative formulations of the cost model should be considered to include other critical factors such as network traffic or server load. In addition, the impact of cooperative caching on overall CDN performance should be further explored.

Thus far in our discussion of the content distribution problem, we have considered the problem from the perspective of a single decision maker, i.e., the CDN provider. In an alternative approach to the content distribution problem, we may consider the problem from the perspectives of both the CDN provider and customer. In this way, the problem can be modeled using a game-theoretic approach. For instance, we may assume that the CDN servers act noncooperatively with the objective of maximizing their individual revenues and, thus, compete with one other to store customers’ content. We may also assume that customers do not cooperate either and attempt to
purchase as much cache space as possible with the objective of maximizing their net benefit.

Given this scenario, the content distribution problem can be modeled as a noncooperative game played among the CDN servers and the customers. With this approach, the optimization problems for the players need to be formulated. For the CDN server problem, if we assume that revenue maximization is the objective, then a revenue function needs to be derived. This objective function may be subject to a set of constraints that ensure demand for cache space is satisfied and that supply is not exceeded. For the CDN customer problem, an objective function needs to be derived that expresses the net benefit gained by the customer as a result of purchasing cache space. This objective may be constrained by the total amount that the customer can invest in caching space. A good starting point for research in this area is a recent work that proposes a game-theoretic approach to the content distribution problem and the request routing problem (Ercetin 2002).

6.1.2. Request Routing. As previously discussed, CDN servers currently use DNS redirection to route requests to the “best” CDN server, where the “best” server is selected based on various proximity measures (e.g., hop count) and/or server feedback information (e.g., number of connections). An important point to emphasize is that DNS redirection relies on a static DNS mapping to make routing decisions. While the frequency with which this mapping is changed varies (recall that the TTL is used to control the lifetime of this mapping), the fact that these mappings are not dynamically changed, in response to various system events, indicates that this approach may result in suboptimal solutions. A more elegant approach would consider the current values for the server selection criteria at run time, and dynamically determine the optimal CDN server. Such an approach should improve QoS, which should decrease the likelihood of SLA violations.

In the request routing problem, we are given a set of ASs, a set of objects, some knowledge of the placement of the objects across the ASs at a particular point in time, and some knowledge of the system state (e.g., server load at each AS, amount of network traffic on each inter-AS link). The problem is then, given a request for an object, to select an AS to serve the object such that the cost to serve the object is minimized. This problem closely resembles the problem of distributed load balancing, which has received considerable attention in the literature (e.g., Karger et al. 1997, Plaxton et al. 1996). In distributed load balancing, the problem is to route requests to a set of servers distributed across a network such that the load on each server is approximately equal. Thus, the distributed load balancing problem has a specific objective, i.e., evenly distributed load, whereas the CDN request routing problem is a more general problem. Approaches to solving distributed load balancing problems are typically based on randomization and hashing techniques (Karger et al. 1997, Plaxton et al. 1996). Given the more general nature of the CDN request routing problem, it appears that optimization techniques are more appropriate. We now present a simple optimization model for this problem to illustrate.

The decision in the request routing problem is which AS to choose to serve the requested object. We can define a set of decision variables \( X_{ij}, \forall i \in I, j \in J \), such that \( X_{ij} = 1 \) if a given request for object \( j \) is routed to AS \( i \), and \( X_{ij} = 0 \) otherwise. A possible objective function is to minimize the total cost to retrieve all objects that are requested at a particular point in time. To compute this cost, we can again let \( o_j \) be the cost to retrieve object \( j \) from the origin server. We can express the cost to retrieve object \( j \) from AS \( i \), in terms of both server and network cost where, for example, server cost may represent load on the server at an AS and network cost may represent latency on the network path used to retrieve the object from the AS. Accordingly, let \( c_i \) be the server cost at AS \( i \), and \( l_{ij} \) be the network cost to retrieve object \( j \) from AS \( i \). Furthermore, let \( r_{ij} \) denote the probability that object \( j \) exists at AS \( i \). Then this objective can be formulated as

\[
\sum_{i \in I} \sum_{j \in J} (r_{ij}(c_i + l_{ij})X_{ij} + (1 - r_{ij})(c_j + l_{ij} + a_j)X_{ij} + a_j(1 - X_{ij})) \quad (2)
\]

In (2), the first term represents the cost to retrieve an object in the case where the request is directed...
to an AS that has the object. The second term in (2) represents the case where the request is directed to an AS that does not have the object and, as a result, additional cost is incurred to retrieve the object from the origin server. The third term in (2) represents the case where the request is routed to the origin server. One set of constraints needed for this particular formulation is a set of structural constraints, of the form \( \sum_{j} X_{ij} = 1, \ i = 1, \ldots, I \), to ensure that only one AS is selected to serve a given request. Additional constraints in this model may include bandwidth constraints and/or server capacity constraints.

Here again, we emphasize that we have presented an overly simplistic formulation of the problem. For example, the objective function assumes linear costs, which is not actually the case for network and server congestion.

As in the case of the content distribution problem, we can also consider both the CDN provider and customer perspectives. Thus, a similar game-theoretic model could be used in which the AS selection decision is made based on each of the players’ objectives (e.g., maximization of net benefit).

An especially challenging issue related to the request routing problem is the need for highly scalable solution techniques. Given the potential size of this problem (e.g., Akamai’s CDN platform has more than 12,000 servers),\(^1\) and the fact that a routing decision must be made for each request (there could be hundreds to thousands of concurrent requests at a given AS), the implementation of efficient solution techniques is critical. Two related design issues must be considered: (1) the type of input data used, and (2) the frequency of solution procedure execution.

For example, historical data may be used to obtain a solution. This approach allows flexibility in terms of the frequency with which the solution procedure must be executed. For instance, the solution procedure could be executed at regular intervals (e.g., daily, weekly). However, a drawback of this approach is that the solution quality depends upon how accurately the historical data predicts the future system behavior. Alternatively, real-time data may be used as input to the solution procedure. This approach has the advantage of providing more accurate solutions, but requires that the solution procedure be executed more frequently. In the extreme case—i.e., to guarantee an optimal solution—the solution procedure must be executed for each request. In practice, however, this will likely not be feasible due to the scale of the problem. Rather, efficient heuristic approaches would need to be developed.

### 6.1.3. Pricing.

As far as we are aware, the problem of determining the optimal pricing scheme for the services offered by a CDN has not yet been addressed in the published literature. For this problem, assume that we are given a set of ASs, a set of objects, the placement of these objects, and the demand for these objects across the ASs. From the perspective of the CDN provider, the pricing problem can be stated as follows: given the placement of and demand for the objects across the ASs, find the price to charge for serving each object via the CDN such that revenue is maximized.

The decisions in this problem are the prices to charge for each of the objects served. We can define a set of decision variables \( P_j, \forall j \in J \), such that \( P_j \) denotes the price charged by the CDN provider to serve object \( j \) via the CDN. A possible objective function is to maximize the total revenue obtained from serving objects via the CDN. To express this revenue, we can define \( d_{ij} \) to be the demand for object \( j \) at AS \( i \), and let \( X_{ij} \) denote the placement of object \( j \) at AS \( i \), as previously defined. Then this objective can be formulated as

\[
\sum_{i \in I} \sum_{j \in J} d_{ij} X_{ij} P_j,
\]

which computes the total revenue based on the known demand for objects and placement of objects. One possible set of constraints for this formulation would ensure that a target revenue is met at each AS location. This set of constraints may be expressed as \( \sum_{j \in J} d_{ij} X_{ij} P_j \geq T_i, \ i = 1, \ldots, I \), where \( T_i \) denotes the target revenue for AS \( i \). A constraint set would be needed in this case to enforce an upper bound on the prices charged. This constraint set may be of the form \( P_j \leq M_j, \ j = 1, \ldots, J \), where \( M \) denotes the maximum price that can be charged for serving an object. The value of \( M \) would be determined from market

conditions reflecting the CDN customers’ willingness to pay.

We emphasize that the above formulation is quite simplistic. For example, this formulation does not take into account the cost incurred when an object must be retrieved from the origin server. Another factor not included in this formulation is QoS. One possible way to incorporate QoS into this formulation would be to add a set of constraints to enforce an upper bound on the average delay experienced by the end user. This upper bound would be determined by the QoS levels specified in the SLAs. Still other constraints, such as supply and demand constraints, may be included in this formulation. Note also that this formulation assumes a usage-based pricing model. Additional formulations could be developed for alternative pricing models, which could then be compared and used to select the appropriate pricing model.

If we also consider the problem from the perspective of the CDN customer, a game-theoretic approach may be used to model this problem. Assume that we are given a set of CDN customers, such that each customer has a fixed amount that it can invest in caching services. Also, assume that each customer has a net benefit function that expresses the benefit gained from the caching services. As previously mentioned, this net benefit function may be based on the average reduction in delay that its end users experience as a result of the caching services. If we assume that this benefit function includes the price as a decision variable, then the problem is to find the price that the customer is willing to pay to maximize its net benefit (note that the resulting solution may be useful in determining the appropriate value for \( M \) previously discussed). This objective could be subject to a set of constraints that ensure a customer does not exceed its maximum investment amount. Once the CDN provider and CDN customer optimization problems are formulated, various properties of the model can be examined, such as whether a Nash equilibrium exists. These and other properties can provide useful insight into the optimal pricing strategy.

The problem described above can be extended to consider multiple levels of QoS. Recall that this problem of pricing for differentiated caching services was addressed in the context of a single IAP in Chuang et al. (2001). The authors mention in Chuang et al. (2001) future plans for extending their model to incorporate pricing strategies for CDNs, an extension that we feel will be a valuable and interesting research work.

6.2. Optimization Problems for Content Generation

Object caching is a commonly used technique to address the performance and scalability problems associated with object-oriented (OO) applications. An open problem related to content generation is the optimal selection of objects to cache in a caching system for OO applications. We now describe this problem.

An OO application can be represented as a set of complex objects \( \{o_1, o_2, \ldots, o_n\} \), where each object \( o_i \) consists of an object hierarchy. Each object in an object hierarchy may be composed of primitive data types or other objects. As an example, consider an online news site. Visitors to the site have the option to create a login profile so that they can receive personalized news. Suppose a visitor to the site, Bob, has entered his login information (user ID and password) through his Web browser. This request invokes a page-generating application at the news site, which generates the page that is returned to Bob. The resulting Web page is shown in Figure 2(a). This page can be thought of as an object, which is itself composed of several presentation-layer objects: Nav Bar, Ad/Offer, Personal News, Current News, and Market Watch. These objects, in turn, may be composed of application-layer objects. For instance, the Current News object is composed of Breaking Stories and Headlines. The Headlines object is further composed of Sports, Weather, Money, and World objects. The Weather object may be further composed of Local Weather and National Weather objects and so on. This object hierarchy is depicted in Figure 2(b).

Given an object hierarchy, one can think of caching decisions at two levels: (1) the design level, and (2) the run-time level. At the design level, the choice to be made is to determine which of the objects should be made “cacheable.” In other words, once an object, say \( o_i \), is “marked” as cacheable at design time, an
instance of $o_j$ would be a candidate for caching by the run-time cache manager.

The other level of cache decision making occurs at run time, when the cache manager decides, on the fly, whether to cache a specific object instance. Also, if the cache manager decides to insert a specific object instance into cache and the cache happens to be full, an additional decision that must be made is what to discard from the cache to make room for the newly arrived instance. The latter problem refers to the cache replacement problem, which has been extensively studied in the literature (see Datta et al. 2003 for a survey of cache replacement policies). The former problem, i.e., the design level problem, to the best of our knowledge, has not yet been addressed in the literature.

One may wonder why we would not simply mark all objects as cacheable. Under this scenario, all caching decision making would be offloaded to the run-time cache manager. As it turns out, this is a nonoptimal strategy for several reasons. One reason is that for any cacheable object, there is a run-time cost to fetch the object from cache, referred to as the cache lookup cost. Each object also has a generation cost, which is the cost to create the object instance.

If the cache lookup cost and generation cost are of the same order, then it is not beneficial to cache the object. Another reason why it is not a good strategy to mark every object as cacheable is that the update rate may be greater than the request rate. Still another reason why it may not be optimal to mark every object as cacheable is that it increases the overhead of the run-time cache manager by increasing its decision space exponentially.

Thus, this design time, object-caching decision problem can be stated as follows. For a set of objects within an application, we are given the hierarchical relationships among the objects, as well as each object’s size, lifetime, access rate, number of instances, and the benefit gained from marking the object as cacheable. The problem is then to identify the objects that should be marked as cacheable such that the total cost of accessing all top-level or root objects from the application layer is minimized. Constraints in this model may include, for instance, performance-related constraints, such as minimum object generation times. A key challenge is to formulate this problem in such a way that it can be efficiently solved. We are currently addressing this problem, and refer interested readers to a preliminary version of this work (Dutta et al. 2002).
6.3. Performance Models of Caching Systems

The work done by OR/MS researchers in performance modeling of caching systems thus far, while useful in establishing a foundation upon which to build for future studies, has considered relatively simplistic systems to keep the models tractable. These models have focused on systems having a single cache. What is needed are analytical models that more closely represent present-day computing architectures from both the networking and server-side perspectives. For example, on the networking side, models are needed that capture the complexities of cooperative caching systems. For example, can the model presented in Bose and Cheng (2000) be extended to include multiple proxies that cooperate to serve requests? It would be extremely valuable to be able to analytically study the different types of cooperative caching architectures discussed in §4.1.

On the server side, models are needed that capture the layered or multitier application architectures that are widely used today. Such models are essential in allowing researchers to accurately characterize the performance impact of caching techniques, such as those discussed in §4.2 on these systems. Figure 3 depicts a high-level queueing model of a multitier application architecture. This figure illustrates the flow of requests through such an architecture (note that the response flow is not shown for the sake of simplicity) and attempts to highlight the nested function calls that typically occur within the application logic. For instance, a presentation layer program, e.g., Active Server Pages (ASP) or Java Server Pages (JSP), may invoke a business logic component (e.g., EJB, COM), which may, in turn, invoke data access software, e.g., Java Database Connectivity or Open Database Connectivity, which requests data from a back-end database system. Queueing would typically occur at each layer. Although not shown in the figure, caching may occur at one or more layers. The result is a complex interaction of software modules, making analytical modeling a challenging problem.

One research work that provides a good starting point for modeling a layered application architecture is Ramesh and Perros (2001). This paper presents a multilayered queueing network model that models a client-server system where clients and servers communicate via synchronous and asynchronous messages. In this model, the servers are grouped such that they form a multilayered structure. The authors analyze this network using a decomposition algorithm.

An ongoing challenge with performance models of caching systems is to achieve a balance between model accuracy and complexity. For instance, a model of a server-side system that incorporates multithreading in the servers would be more accurate than one that does not. The queueing model presented in Ramesh and Perros (2001) can be used to model mul-

---

**Figure 3** High-Level Queueing Model of a Multitier Application Architecture

[Diagram of a multitier application architecture showing Web Server Tier, Application Server Tier, and Back-end Systems Tier with various components like Apache, ASP, JSP, COM, EJB, JDBC, ODBC, File Server, DB Server, etc.]
7. Conclusion

As Internet traffic continues to rapidly grow, infrastructure capacity is also increasing. The demand for use is outpacing the infrastructure capacity, however, resulting in poor quality service. A promising approach to help alleviate this problem is the use of caching techniques. In this paper, we have identified the main causes of latency on the Web, and have discussed a variety of caching strategies that help reduce these sources of latency. We have also surveyed the OR/MS work related to WWW caching. As our survey indicates, there appears to be an increasing amount of interest in WWW caching from the OR/MS research community. In addition, we have presented a number of open OR/MS research problems, which we hope will attract an even greater number of OR/MS researchers and practitioners to this challenging field.

References


2 A supplemental appendix to this paper is available at mansci.electronic.companion.html.


Gamma, E., R. Helm, R. Johnson, J. Vliss. 1994. Design Pattern Elements of Reusable Object-Oriented Software. Addison Wesley Professional Computing Series, Boston, MA.


2003. Mirror Image Internet InstaContent, Microsoft.


Accepted by Arthur M. Geoffriau and Ramaga Krishna; received January 2002. This paper was with the authors 19 months for 5 revisions.